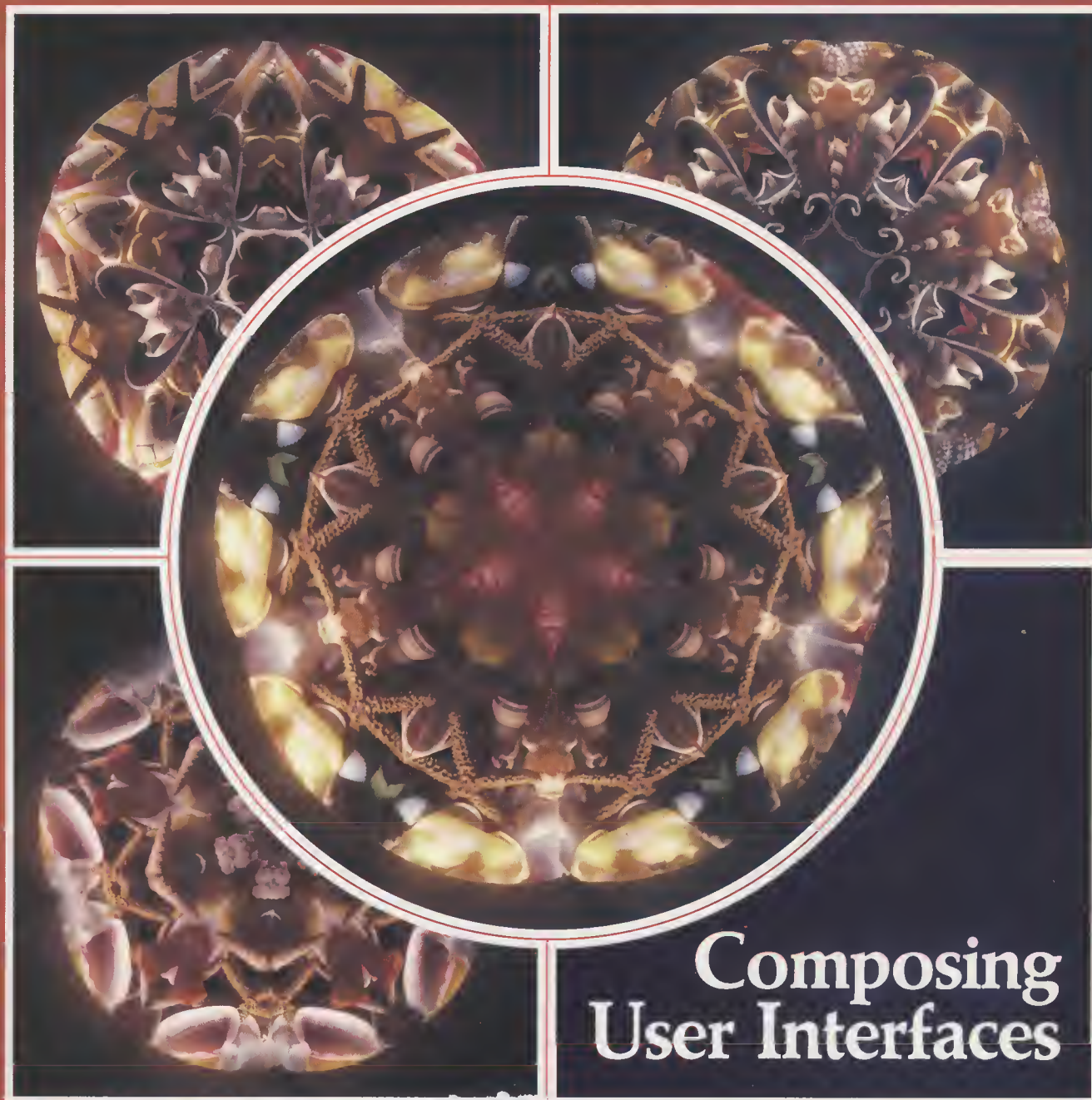


FEBRUARY 1989

COMPUTER



Composing User Interfaces

Also in this issue:

Interconnection Networks
Code Optimization
Ada Compiler Technology
Computing as a Discipline
A Discipline Matures



IEEE COMPUTER SOCIETY



THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.



IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN

ICCAD-89

sponsored by
IEEE CIRCUITS AND SYSTEMS SOCIETY
THE IEEE COMPUTER SOCIETY

A CONFERENCE FOR THE
EE CAD PROFESSIONAL

NOVEMBER 6 - 9, 1989
SANTA CLARA, CA

CALL FOR PAPERS

The 1989 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN will be held November 6 - 9, 1989. The Conference oriented towards Electrical Engineering CAD professionals, concentrating on CAD for Electronic Circuit Design.

AREAS OF INTEREST

Original technical papers on (but not limited to) the following topics are invited:

- 1) **HIGH LEVEL SIMULATION:** Functional, Behavioral, Architectural, Mixed-mode, Fault Simulation, Hardware Acceleration and Parallel Techniques for High Level Simulation
- 2) **SIMULATION:** Timing, Circuit, Device and Process Simulation, Modeling, Hardware Acceleration and Parallel Techniques for Simulation
- 3) **PLACEMENT AND FLOORPLANNING:** Placement, Floorplanning, Area Estimation, Layout Systems, Hardware Acceleration and Parallel Techniques
- 4) **ROUTING:** Routing algorithms for IC and System Design
- 5) **LAYOUT VERIFICATION/EXTRACTION:** Circuit Extraction/Verification, Design Rule Checking and Electrical Rule Checking, Symbolic Design and Compaction
- 6) **TESTING:** Design for Testability, Automatic Test Pattern Generation, Built-in Test
- 7) **LOGIC AND ANALOG SYNTHESIS:** Logic Synthesis and Formal Verification, Technology Mapping, Finite-State Machine Synthesis, Analog Circuit synthesis
- 8) **HIGH LEVEL SYNTHESIS:** Functional, Behavioral, Architectural Synthesis, Synthesis Systems, Silicon Compilation
- 9) **CAD FRAMEWORKS:** Tool Integration, User-interfaces, Data-bases for IC and system design.

AUTHOR INFORMATION

Authors should submit 12 COPIES of:

- 1) a one-paragraph abstract.
- 2) a more detailed description not to exceed 18 double-spaced pages, figures and tables included. Excessively long submissions and previously published papers will be returned to the authors.

FORMAT:

The ONE-PARAGRAPH ABSTRACT, typed on one separate page, should clearly and precisely state what is new and point out the significant results. Succinctness is required since this paragraph may be included in the Advance Program.

In the detailed description, the author must objectively address why the proposed contribution is superior to prior work or what the significance of the contribution is, if breaking new ground. Demonstration of superiority in algorithms and strategies with heuristics is required through a description of the programming implementation and application to "real" problems. Additional mathematical proofs are welcome. The contribution should address an area of current technical interest to the CAD professional. A clear description of the new contribution, status of the work and significant examples and results should be given.

COVER PAGE REQUIREMENTS:

Submissions should include, on the cover page: the title of the paper; the category which most clearly matches the paper's content (see Areas of Interest); the full name, complete return address, telephone number and affiliation of each author; and clear identification of the individual to whom all communication should be addressed. In giving your return address, please consider that the communications for paper acceptance and mailing of the author's kit occur in the month of July.

AUTHOR'S SCHEDULE

Deadline for submissions:	April 28, 1989
Notification of acceptance:	June 30, 1989
Deadline for final version:	August 11, 1989

SEND TO:

ICCAD-89 Secretary
IEEE Computer Society
1730 Massachusetts Ave. N.W.
Washington, D.C. 20036-1903
Telephone: (202) 371-1013

GENERAL CHAIRPERSON

Andrzej J. Strojwas
ECE Department
Carnegie-Mellon University
Pittsburgh, PA 15213
(412) 268-3530

PROGRAM CHAIRPERSON

Alberto Sangiovanni-Vincentelli
Department of EE and CS
University of California, Berkeley
Berkeley, CA 94720
(415) 642-4882

PUBLICATIONS CHAIRPERSON

Satoshi Goto
C and C Information Technology Research Labs
NEC Corporation
1-1 Miyazaki 4-Chome
Miyamae-ku, Kawasaki
Kanagawa-213, Japan
011-81-44-856-2170

NEW FOR TELECOMMUNICATION ANALYSTS

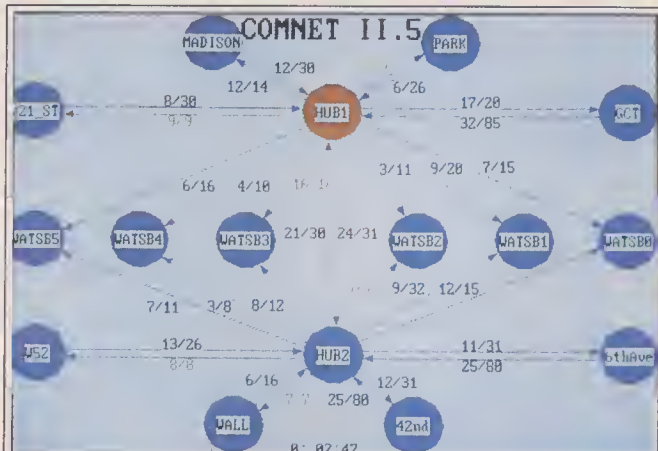
BEFORE

SIMPLIFYING ASSUMPTIONS

- Use only exponential service & interarrival times
- Assume independent links for end-to-end performance measures
- Ignore effects of adaptive routing
- Compute only steady-state averages

Unrealistic simplifications of analytical methods--costs and delays of programming

AFTER



Realistic simulation of your telecommunication network--quick results, no programming

See your proposed circuit or packet-switched network perform
COMNET II.5 now gives you easy-to-understand network
simulation results quickly--no programming

With COMNET II.5 you simply describe your telecommunication network and its routing algorithms.

Simulation follows immediately--no programming delays.

Easy-to-understand results

You get an animated picture of your network. Routing choices and changing levels of network utilization during the simulation are apparent.

Your reports show blocking probabilities, call queueing and packet delays, network throughput, circuit group utilization, and circuit group queue statistics.

Animation increases everyone's understanding of the network operation and builds confidence in your analysis.

Your network simulated

You can analyze any wide-area network which uses circuit switching, message switching, or packet switching. Virtual-circuit or data-group operation can be modeled.

Various alternate-routing and adaptive shortest-path routing algorithms are built-in.

Free trial and training offer

The free trial contains everything you need to try COMNET II.5™ on your computer.

You may develop your own network or modify one of ours.

Try the COMNET II.5 approach, the quality and timeliness of our support, the accuracy of our documentation, and the facilities for error checking--**everything you need for a successful project.**

No cost, no obligation.

Act now--free training offer

For a limited time we also include free training. Typical applications are demonstrated.

Call today to avoid disappointment--class size is limited.

For immediate information

Call Kelly Cox at (619) 457-9681, FAX (619) 457-1184. In the UK, call Richard Eve on (01) 528-7980.

The quickest way for you to evaluate telecommunication network alternatives is with COMNET II.5.

COMNET II.5 results are realistic--the drastic simplifying assumptions required by analytical methods are eliminated.

Rush information on the COMNET II.5 free trial and training offer

Free trial--see for yourself how COMNET II.5 quickly answers telecommunication network performance questions by eliminating programming.

Limited offer--Act now for free training.

☐ Send information on your Special University Offer.

Name

Organization

Address

City State Zip

Telephone Computer

Return to: IEEE COMP

CACI Products Company
3344 North Torrey Pines Court
La Jolla, California 92037
Call Kelly Cox at (619) 457-9681. FAX (619) 457-1184.

In the UK:
CACI Products Division
Regent House, 89 Kingsway
London WC2B 6RH, United Kingdom
Call Richard Eve on (01) 528-7980.

COMPUTER

February 1989

Published by the IEEE Computer Society

Vol. 22 No. 2

ARTICLES

8 Composing User Interfaces with InterViews

Mark A. Linton, John M. Vlissides, and Paul R. Calder

The InterViews toolkit offers a rich set of composition mechanisms and a variety of predefined objects, allowing easy implementation of complex user interfaces.

25 Performance of Multiprocessor Interconnection Networks

Laxmi N. Bhuyan, Qing Yang, and Dharma P. Agrawal

Multiprocessor designers need analytical techniques to evaluate network performance. This article presents a tutorial on these evaluation tools to guide designers through the design process.

42 Code Optimization Across Procedures

Stephen Richardson and Mahadevan Ganapathi

Procedure calls can be a major obstacle to the analysis of computer programs, preventing significant improvements in program speed. Can this obstacle be overcome?

52 Issues in Ada Compiler Technology

Mahadevan Ganapathi and Geoffrey O. Mendal

Compiler technology is inadequate to support many Ada features, but — an encouraging note — the speed of Ada compilers has doubled from the first to the second generation.

SPECIAL REPORTS

63 Computing as a Discipline

Peter J. Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young

Final Report of the ACM Task Force on the Core of Computer Science, in cooperation with the IEEE Computer Society.

72 A Discipline Matures

David Gries, Terry Walker, and Paul Young

No single entity represents computing research to the public and to policy makers. As a result of Snowbird 88, the Computing Research Board plans to fill that role.

Circulation: *Computer* (ISSN 0018-9162) is published monthly by the IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720, phone (714) 821-8380. IEEE Computer Society Headquarters, 1730 Massachusetts Ave. NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription included in society member dues. Nonmember subscription rates: available upon request. Single copy prices: members \$10.00; nonmembers \$20.00. This magazine is also available in microfiche form.

Postmaster: Send undelivered copies and address changes to *Computer*, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Second class postage is paid at New York, New York, and at additional mailing offices.

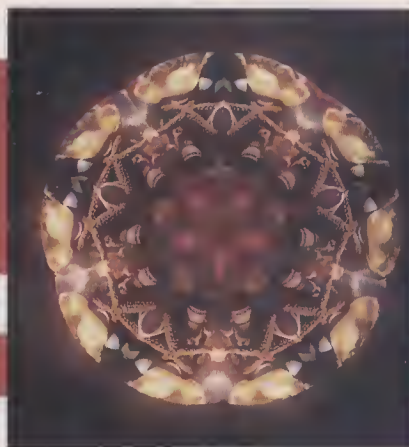
Copyright and reprint permission: Copyright © 1989 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US copyright law for private use of patrons: (1) those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 29 Congress Street, Salem, MA 01970; (2) pre-1978 articles without fee. Instructors are permitted to photocopy isolated articles for noncommercial classroom use without fee. For other copying, reprint, or republication permission, write to Permissions Editor, *Computer*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *Computer* does not necessarily constitute endorsement by the IEEE or the Computer Society. All submissions are subject to editing for style, clarity, and space considerations. *Computer* subscribes to the Computer Press Association's code of professional ethics.



DEPARTMENTS

- 4 Letters to the Editor**
P68000 μ Lab — another view
- 76 The Open Channel**
- 78 Standards**
Rugged Bus, Futurebus groups join forces
- 80 Computer Society News**
Call for nominations, 1989 election timetable;
CS membership passes 100,000 milestone
- 83 Update**
- 87 Product Reviews**
A Turbo family portrait
- 94 New Products**
- 100 IC Announcements**
- 102 Microsystem Announcements**
- 104 Conferences**
Visual motion; AI applications; databases
- 107 Calendar**
- 111 Call for Papers**
- 124 Book Reviews**
- 127 New Literature**



Cover photos: Peach Reynolds, KaleidoPhotography, Austin, Texas, creates kaleidoscopic patterns — here, of seashells — with a system of optical-grade mirrors. The cover photos were taken using a Nikon FE2 equipped with a 55mm Nikkor Macro lens with an 80-b filter at f-3.5, 1/60 second on KodaChrome 64 PKR film.

Cover design: Jay Simpson,
Design & Direction

In the next issue

Robotics and Automation

Computer Society Information Page, inside back cover; Membership Application, 113; Career Opportunities, 114; Change-of-Address Form, 70; Advertiser/Product Index, 128; Reader Service Card, 128A

COMPUTER

10662 Los Vaqueros Circle, Los Alamitos, CA 90720

Editor-in-Chief

Bruce D. Shriver, University of Hawaii

Editorial Board

Dharma P. Agrawal, North Carolina State University

Dennis R. Allison, Stanford University

Scott Davidson, AT&T

Richard Eckhouse, MOCO

Scott E. Fahlman, Carnegie Mellon University

Gideon Frieder, Syracuse University

Edmund Gallizzi, Eckerd College

John Gurd, University of Manchester

James H. Haynes, University of California, Santa Cruz

Wiley McKinzie, Rochester Institute of Technology

K.M. Mohiuddin, IBM Almaden Research Laboratory

Yale Patt, University of Michigan

William Riddle, Software Productivity Consortium

Sallie Sheppard, Texas A&M University

Helen M. Wood, National Oceanic and Atmospheric Administration

Akinori Yonezawa, Tokyo Institute of Technology

Submissions

Send six double-spaced copies of articles and special-issue proposals to Bruce D. Shriver, Dept. of Decision Sciences, University of Hawaii, 2404 Maile Way, Honolulu, HI 96822, phone (808) 948-7231. For electronic submission, Shriver's addresses are b.shriver on Compmail + , shriver@uhccux on Bitnet, and shriver@uhccux.uhcc.hawaii.edu.

Staff

Editor and Publisher: True Seaborn

Assistant Publisher: Douglas L. Combs

Assistant to the Publisher: Pat Paulsen

Managing Editor: Marilyn Potes

Assistant Editors: Nancy Hays,
Chuck Governale,
Steve H. Wilcox

Art Director: Jay Simpson

Design/Production: Miriam Wiegel

Membership/Circulation: Christina Champion

Advertising Director: Dawn Peck

Advertising Coordinators: Heidi Rex, Marian Tibayan

Magazine Advisory Committee

Sallie Sheppard (acting chair), James H. Aylor, Victor Basili, Michael Evangelist, Joe Hootman, Ted Lewis, David Pessel, True Seaborn, Bruce D. Shriver, John Staudhammer

Publications Board

Sallie Sheppard (chair), James H. Aylor, Victor Basili, J. Richard Burke, Jon T. Butler, J.T. Cain, Sumit DasGupta, Michael Evangelist, Joe Hootman, Glen G. Langdon, Ted Lewis, Ming T. Liu, Theo Pavlidis, David Pessel, Bruce D. Shriver, Charles B. Silio, John Staudhammer, Harold Stone, Steven L. Tanimoto

LETTERS TO THE EDITOR

P68000 μ Lab — another view

To the Editor:

I am writing in reference to the review "The P68000 μ Lab," by D. Noah Eckhouse, which appeared in the December 1988 issue of *Computer* (p. 90).

We have used the P68000 μ Lab for a number of years, as have other schools throughout the US and Canada. This particular educational trainer, like others we have used (e.g., Heath ET-3400, Microtech's Micrprofessor, and National's SC/MP Trainer), can be used in or out of an educational environment but typically not without supplemental materials. Although Heath does provide very good educational materials that can be used with the ET-3400, they are not used at all of the levels at which the ET-3400 is used.

Teaching tools such as the μ Lab are used at a variety of levels and for multiple purposes throughout two- and four-year schools for computer science, tech-

nology, and engineering. Since no one coverage is universal, the material supplied with any such trainer must be supplemented with appropriately chosen materials.

A course announcement in the same issue of *Computer* (p. 85) indicates courses that use the μ Lab and supplement the hardware with appropriate materials.

Two books are currently being written that represent the type of learning materials Eckhouse indicates he needs in addition to or in place of the references included in the μ Lab User's Manual. These would be useful as would other texts for learning the 68000 microprocessor.

After reading the review, I paid particular attention to the advertisements regarding the μ Lab and could find no indication or suggestion of documentation other than the (1) Programmer's Reference Manual and (2) P68000 μ Lab User's Manual.

I am in complete agreement with Eckhouse's comments regarding (1) the technical quality ("very well done"), the User's Manual description ("fine"), and the hardware ("well done").

It seems a little strong to apply the term "lousy documentation" (which Eckhouse contradicts in reference to the User's Manual with "...their description is fine"), in view of the fact this is an *excellent* tool for teaching both in and out of an educational environment *supplemented* with the materials at your level of understanding, which only the user can identify for certain.

Marlin H. Mickle
University of Pittsburgh

Thank you for your letter; I appreciate your taking the time to write and offer a different viewpoint.

Richard H. Eckhouse, Jr.
Product Reviews Editor

State Sponsored Chair and Professor in Computer Science

NJIT invites nominations for and applications from distinguished researchers for the New Jersey State Sponsored Chair in Computer Science. The candidate will be expected to provide strong research leadership and pivotal work in any areas of integrated systems such as software engineering, distributed knowledge base systems, computer communications and networking, database systems and languages, and distributed computing. Candidates should be nationally and internationally recognized for their contributions and should have proven academic/industrial experience. He/she can expect a competitive salary for a distinguished professorship plus a research budget commensurate with responsibilities.

Computer and Information Science is a rapidly expanding department with the largest program in NJ, and offers B.S., M.S., and Ph.D. programs. CIS has extensive laboratory facilities offering excellent potential for industrial support.

NJIT is the public technological university of New Jersey, with nearly 8,000 students enrolled in baccalaureate through doctoral programs in four colleges: Newark College of Engineering, the School of Architecture, the School of Industrial Management and College of Science and Liberal Arts.

NJIT does not discriminate on the basis of sex, race, color, religion, national or ethnic origin, or age in employment.

Send resume: Personnel Box CIS.



**New Jersey
Institute of Technology**

Newark, New Jersey 07102

Technische Hochschule Darmstadt

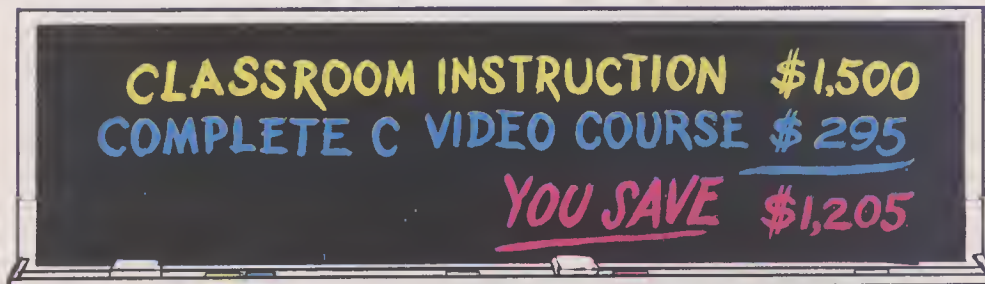


TH Darmstadt Computer Science and Computer Engineering

The Technical University of Darmstadt in Germany invites applications and nominations for the position of Professor(C4) in Computer Systems (Rechnersysteme Nachfolge Prof. Piloty Kenn-Nr.: P 70). Candidates for the position should have a well established reputation in research and experience in teaching in the field of computer science and computer engineering. Duties of the Professor include teaching at the undergraduate and the graduate level especially in logic design, computer organization and computer architecture.

Candidates are invited to submit their application together with a curriculum vitae, a list of publications, references and a description of their research activities to the President of THD, Karolinenplatz 5, D-6100 Darmstadt, West Germany.

Take Our Course In C And The First Lesson You'll Learn Is In Economics.



C's power and portability make it the language of choice for software developers. Unfortunately, learning C can be a very costly proposition. Classroom instruction is, in a word, expensive. And many C video courses carry hefty price tags.

The top C video course at the lowest possible price

But now, there's The Complete C Video Course from Zortech. It's the ultimate C training tool for home or work. And all it costs is \$295.



You get ten videos with 36 lessons covering all levels of programming skill. A comprehensive, easy-to-follow 365 page workbook. And even a free C compiler.

Free C compiler included

Yes, that's right. The Complete C Video Course includes our famous C compiler (it runs on any MS-DOS machine) with linker, library manager, full graphics library and on-line help. It's the choice of professional programmers everywhere for fast code, fast development and fast debugging.

```
#include <stdio.h>
#define NAMLEN 15
#define NUMMARK 4
struct person
{
    char name[NAMLEN];
    int mark[NUMMARK];
};
```

Learn C in as little as two weeks

Speaking of speedy, with The Complete C Video Course you can learn C in only two weeks.

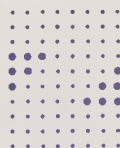
Compare that with the up to four months it can take to learn C in class.

Each lesson averages 17 minutes of clear, concise instructions. Used in conjunction with our workbook you'll find they provide everything you need to know to become

proficient in programming in C.

Save your company thousands

If you think The Complete C Video Course is a great way for you to save money learning C, think about how much it could save your company. Use it instead of sending programmers to school and you'll save thousands. What's more, The Complete C Video Course is even tax deductible. C is unquestionably the most valuable programming language you can master. And now you can get everything you need to become productive in it from course to compiler to tools for an economical \$295. Mail the coupon or call our hotline to receive it ASAP.



ZORTECH

Look at all these C video pluses

- ★ Only \$295 complete.
- ★ Ten videos with 36 lessons.
- ★ Comprehensive 365-page workbook.
- ★ Free C compiler with linker, library manager, full graphics library and on-line help.
- ★ Compiler and hardware independent.
- ★ Designed to help you learn C in as little as two weeks.
- ★ Tax deductible.



Zortech Inc.
366 Massachusetts Ave.
Arlington, MA, 02174
Tel: (617) 646-6703
Fax: (617) 643-7969

- ★ Yes, rush me The Complete C Video Course including free C compiler for \$295.00 (VHS only)
- ★ Please include (No.) extra workbooks at \$29.95 each.
- ★ I'd like to order (No.) extra C compilers with this course at the special price of \$49.95.

Name/Company

Address

Phone

City

State Zip

Here's my check for

VISA/MC #.

Exp. Date.

% 2 Prices do not include shipping



The Complete C Video Course \$295

Order Hotline (800)848-8408

Reader Service Number 1



Clean Elegant Powerful...

... *Eiffel*

See us at
**SOFTWARE
DEVELOPMENT '89**

"Booths 106, 108, 206, 207"
and at

UniForm 1989
"Booth 2807"

Judge us by the companies we keep:

AT&T

Boeing

GTE

Sun Microsystems

Hewlett-Packard

Lawrence Livermore Nat. Lab.

Bell Northern Research

Cognos

John Fluke Manufacturing Co.

Philips

Tektronix

Sandia National Lab...

We thank these companies and many others around the world for selecting Eiffel, the object-oriented language and environment of the future. These users have already applied Eiffel to an amazing array of applications: real-time simulation, fourth-generation language development,

graphics, user interface management, compiler writing, editors, CASE tool development, and many more.

If you are in the business of producing leading-edge software, shouldn't you be one of the companies we keep?

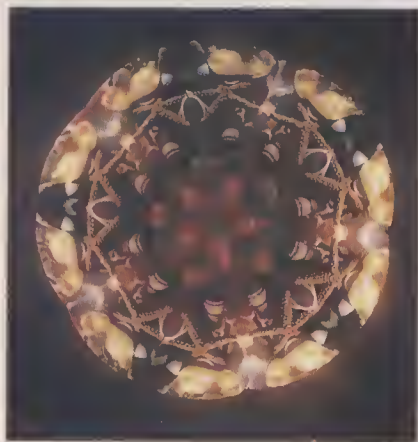
Interactive Software Engineering Inc.

270 Storke Road, Suite 7, Goleta, CA 93117 USA
Telephone: (805) 685-1006 FAX: (805) 685-6869



European Distributor: S.O.L., 4, rue R. Barthélemy 92120 Montrouge France, Phone: (33) (1) 46.57.13.36. Fax: (33) (1) 46.57.01.03

Eiffel is a trademark of Interactive Software Engineering Inc.



Composing User Interfaces with InterViews

Mark A. Linton, John M. Vlissides, and Paul R. Calder
Stanford University

Graphical user interfaces for workstation applications are inherently difficult to build without abstractions that simplify the implementation process. To help programmers create such interfaces, we considered the following questions: What sort of interfaces should be supported? What constitutes a good set of programming abstractions for building such interfaces? How does a programmer build an interface given these abstractions? Practical experience has guided our efforts to develop user interface tools that address these questions. We make the following observations:

All user interfaces need not look alike. It is desirable to maintain a consistent "look and feel" across applications, but users often have different preferences. For example, one user may prefer pop-up menus, while another insists on pull-down menus. Our tools must therefore allow a broad range of interface styles and must be customizable on a per-user basis.

User interfaces need not be purely graphical. Many application designers prefer iconic interfaces because they believe

The InterViews toolkit offers a rich set of composition mechanisms and a variety of predefined objects, allowing easy implementation of complex user interfaces.

novices understand pictures more readily than text. However, recent work¹ suggests that excessive use of icons can confuse the user with unfamiliar symbolism. A textual interface might be more appropriate in a given context. The choice of graphical or

textual representation should favor the clearest alternative.

User interface code should be object-oriented. Objects are natural for representing the elements of a user interface and supporting their direct manipulation. Objects provide a good abstraction mechanism, encapsulating state and operations, and inheritance makes extension easy. Our experience shows that, compared with a procedural implementation, user interfaces written in an object-oriented language are significantly easier to develop and maintain.

Interactive and abstract objects should be separate. Separating user interface and application code makes it possible to change the interface without modifying the underlying functionality, and vice versa. This separation also facilitates customization by allowing several interfaces to the same application. It is important to distinguish between interactive objects, which implement the interface, and abstract objects, which implement operations on the data underlying the interface.

An effective way to support these principles is to equip programmers with a

toolkit of primitive user interface objects that use a common protocol to define their behavior. The protocol allows uniform treatment of user interface objects, enabling in turn the introduction of objects that compose primitives into complete interfaces. Different classes of composition objects can provide different sorts of composition. For example, one class of composition object might arrange its components in abutting or *tilled* layouts, while another allows them to overlap in prescribed ways. A rich set of primitive and composition objects promotes flexibility, while composition itself represents a powerful way to specify sophisticated and diverse interfaces.

Composition mechanisms are central to the design of InterViews, a graphical user interface toolkit we have developed at Stanford. InterViews is a library of C++² classes that define common interactive objects and composition strategies. Figure 1 depicts how objects from the InterViews library are incorporated into an application, and Figure 2 shows the relationship between the various layers of software that support the application. Primitive and composition objects from the library are linked into application code. The window system is entirely abstracted from the application; the application's user interface is defined in terms of InterViews objects, which communicate with the window and operating systems.

InterViews supports composition of three object categories. Each category is implemented as a hierarchy of object classes derived from a common base class. Composition subclasses within each class hierarchy allow hierarchical composition of object instances.

(1) Interactive objects such as buttons and menus are derived from the *interactor* base class. Interactors are composed by scenes; scene subclasses define specific composition semantics such as tiling or overlapping.

(2) Structured graphics objects such as circles and polygons are derived from the *graphic* base class. Graphic objects are composed by pictures, which provide a common coordinate system and graphical context for their components.

(3) Structured text objects such as words and whitespace are derived from the *text* base class. Text objects are composed by clauses; clause subclasses define common strategies for arranging components to fill available space.

The base classes define the communication protocol for all objects in the hierar-

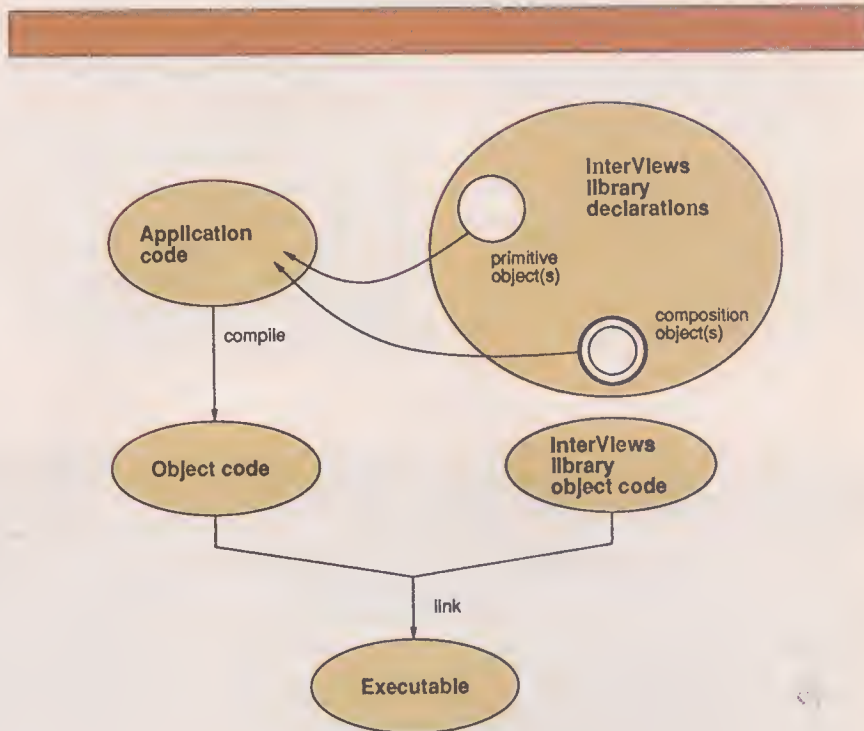


Figure 1. Incorporating InterViews objects into an application.

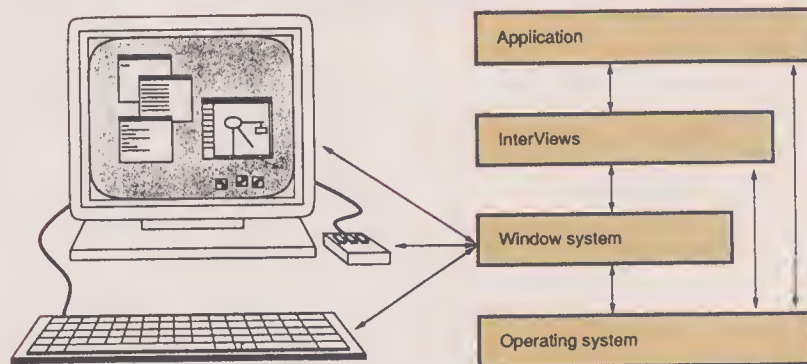


Figure 2. Layers of software underlying an application.

chy. The composition classes define the additional protocol needed by the elements in a composition, such as operations for inserting and removing elements and for propagating information through the composition (see the sidebar entitled "Primitive and composition protocols").

Hierarchical composition gives the programmer considerable flexibility. Com-

plex behavior can be specified by building compositions that combine simple behavior. The composition protocol facilitates the tasks of both the designer of a user interface toolkit and the implementor of a particular user interface. The toolkit designer can concentrate on implementing the behavior of a specific component in isolation; the interface designer is free to

combine components in any way that suits the application.

In this article we focus on using InterViews to build user interfaces. We present several simple applications and show how InterViews objects can implement their interfaces. We also illustrate the benefits of separating interactive behavior and abstract data in several different contexts. Finally, we discuss InterViews support for end-user customization as well as the status of the current implementation.

Interactor composition

An interactor manages some area of potential input and output on a workstation display. A scene composes a collection of one or more interactors. Because a scene is itself an interactor, it must distribute its input and output area among its components. In this section, we discuss the various InterViews scene subclasses that provide tiling, overlapping, stacking, and encapsulation of components. We concen-

trate on how these scenes are used rather than giving their precise definitions.

Boxes and glue. Consider the simple dialog box shown in Figure 3. It consists of a string of text, a button containing text, and a white rectangular background surrounded by a black outline. Pushing the button will cause the dialog box to disappear. The dialog box will maintain a reasonable appearance when a window manager resizes it. If parts of the dialog

Primitive and composition protocols

We can think of the set of operations defined on an object as a communication protocol that the object understands. Since objects cannot access the internal state of other objects, interobject dependencies are limited by the semantics of the protocol. Objects are thus isolated from one another, promoting modularity and reusability. Furthermore, objects derived from a common base class (thus obeying a common protocol) can be used without knowledge of their specific class; operations redefined by the subclass are automatically invoked on the objects instead of the corresponding base class operations (a form of dynamic binding). A common protocol allows composition objects to treat their components uniformly. Dynamic binding lets composition objects take advantage of subclass-specific behavior without modification. Together, these attributes make composition possible.

Interactor protocol. The protocol for interactors includes the following operations:

- void Draw();
- void Redraw(Coord left, Coord bottom, Coord right, Coord top);
- void Resize();
- void Update();
- void Handle(Event&);
- void Read(Event&);

The Draw operation defines the interactor's appearance. A call to Draw causes the interactor to draw itself in its entirety. Redraw is called whenever a part of an interactor needs to be redrawn (for example, when it had been obscured but is now visible). A call to Resize notifies the interactor that the screen space it occupies has changed size. The interactor can then take whatever action is appropriate. Draw, Redraw, and Resize are automatically called by InterViews library code in response to window system requests. The Update operation indicates that some state on which the interactor depends may have changed; the interactor will usually Draw itself in response to an Update call. Typically, when a subject changes it will call Update on its views.

Interactors handle input events with the Handle operation. Each event is targeted to a particular interactor. Any interactor can Read the next event from the global event queue. Handle and Read can be used to create event-driven input handling, in which only one interactor is responsible for reading events and forwarding them to their target.

Scene protocol. Scenes add several operations for component management to the basic interactor protocol:

- void Insert(Interactor*);
- void Insert(Interactor*, Coord x, Coord y, Alignment);
- void Remove(Interactor*);
- void Raise(Interactor*);
- void Move(Interactor*, Coord x, Coord y, Alignment);
- void Change(Interactor*);
- void Propagate(boolean);

Insert and Remove specify a scene's components. Raise modifies the front-to-back ordering of components within a scene to bring the specified component to the top. Move suggests a change in the position of a component within the scene. Not all scenes implement all these operations. For instance, it does not make sense to call Raise on a monoscene, since it can have only one component.

The Change operation tells a scene that one of its components has changed. A scene can do one of two things in response to a Change: It can propagate the change by calling Change on its parent, or it can simply reallocate its components' screen space. The Propagate operation specifies which behavior is required for a particular instance.

Graphic protocol. The graphic base class defines the protocol for drawing objects, manipulating graphics state, and hit detection. Operations include:

- void Draw(Canvas*);
- void DrawClipped(Canvas*, Coord, Coord, Coord, Coord);
- void Erase(Canvas*);
- void EraseClipped(Canvas*, Coord, Coord, Coord, Coord);
- void SetColors(PColor* f, PColor* b);
- void SetPattern(PPattern*);
- void SetBrush(PBrush*);
- void SetFont(PFont*);
- void Translate(float dx, float dy);
- void Scale(float sx, float sy, float ctrx =0.0, float ctry =0.0);
- void Rotate(float angle, float ctrx =0.0, float ctry =0.0);
- void SetTransformer(Transformer*);
- void GetBounds(float&, float&, float&, float&);
- boolean Contains(PointObj&);
- boolean Intersects(BoxObj&);

box previously covered by other windows are exposed, then the newly exposed regions will be redrawn.

InterViews provides abstractions that closely model the elements, semantics, and behavior of the dialog box. A user interface programmer can express the implementation of the interface in the same terms as its specification. The InterViews library contains a variety of predefined interface components. In the dialog box, we will use *message*, *push button*, *box*,

glue, and *frame*. (See the sidebar entitled "Glossary" for definitions of these terms.)

We use boxes and glue to compose the other elements of the dialog box. The composition model is a simplified version of the TeX³ boxes and glue model. This model makes it unnecessary to specify the exact placement of elements in the interface, and it eliminates the need to implement resize behavior explicitly.

Two types of box are used: An hbox tiles its components horizontally, while a vbox



Figure 3. A simple dialog box.

In addition to the operations for setting graphics state attributes and coordinate transformations, there are complementary operations for obtaining the current values of these parameters. The *Contains* and *Intersects* operations determine whether a user clicked on a graphic. *PointObj* and *BoxObj* specify a point and a rectangular region, respectively. *Contains* can detect an exact hit on a graphic; *Intersects* can detect a hit within a certain tolerance.

Picture protocol. Each picture maintains a list of component graphics. A picture draws itself by drawing each component with a graphics state formed by concatenating the component's state with its own. Pictures define default semantics for concatenation; subclasses of picture can redefine the semantics or rely on their components to do the concatenation.

Contains, *Intersects*, and bounding box operations defined in the graphic base class are redefined in the picture class to consider all the components relative to the picture's coordinate system. The picture class defines operations for editing and traversing its list of components. Pictures also define operations for selecting graphics they compose based on position:

- `Graphic * FirstGraphicContaining(PointObj&);`
- `Graphic * FirstGraphicIntersecting(BoxObj&);`
- `Graphic * FirstGraphicWithin(BoxObj&);`
- `Graphic * LastGraphicContaining(PointObj&);`
- `Graphic * LastGraphicIntersecting(BoxObj&);`
- `Graphic * LastGraphicWithin(BoxObj&);`
- `int GraphicsContaining(PointObj&, Graphic**&);`
- `int GraphicsIntersecting(BoxObj&, Graphic**&);`
- `int GraphicsWithin(BoxObj&, Graphic**&);`

The `...Containing` operations return the graphics containing a point; `...Intersecting` operations return the graphics intersecting a rectangle; `...Within` operations return the graphics falling completely within a rectangle.

Pictures draw their components starting from the first component in the list. The `Last...` operations can select the "topmost" graphic in the picture, while `First...` operations select the "bottommost."

Text protocol. The text object protocol includes the following operations:

- `void Draw(Layout*);`
- `void Locate(Coord &x1, Coord &y1, Coord &x2, Coord &y2);`
- `void Reshape( );`

Draw defines the appearance of an object in a given layout. A *Layout* object defines the area of the screen into which a hierarchy of text objects will be composed. *Locate* is used for hit detection on text objects. *Reshape* calculates geometric information about an object for use in implementing composition strategies.

Clause protocol. Clauses add operations for stepping through components and for modifying the list of components:

- `Text * First( );`
- `Text * Succ(Text*);`
- `Text * Pred(Text*);`
- `boolean Follows(Text *, Text*);`
- `void Append(Text*);`
- `void Prepend(Text*);`
- `void InsertAfter(Text * old, Text*);`
- `void InsertBefore(Text * old, Text*);`
- `void Replace(Text * old, Text*);`
- `void Remove(Text*);`

First returns the leftmost or topmost component. *Succ* and *Pred* return the successor or predecessor of a component. *Follows* can determine if one component comes before or after another.

To probe further. We have only considered the basic elements of the various protocols in this discussion. A more detailed look at these protocols and the implementations behind them can be found elsewhere.^{1,2}

References

1. M.A. Linton, P.R. Calder, and J.M. Vlissides, *InterViews: A C++ Graphical Interface Toolkit*, Tech. Report CSL-TR-88-358, Stanford Univ., Stanford, Calif., July 1988.
2. J.M. Vlissides and M.A. Linton, "Applying Object-Oriented Design to Structured Graphics," *Proc. 1988 USENIX C++ Conf.*, Oct. 1988, pp. 81-94; also available as Tech. Report CSL-TR-88-364, Stanford Univ., Stanford, Calif., Aug. 1988.

Glossary

box, hbox, vbox Scenes that support tiled composition of interactors.

button The button base class defines the interface to generic button interfaces. *Push buttons* provide a momentary contact interface. *Radio buttons* allow the user to select one of several mutually exclusive choices.

button state A subject that maintains state associated with one or more buttons.

clause The base class for structured text composition objects.

deck A scene that stacks interactors.

display A clause that defines an indented text layout.

frames Monoscenes that embellish their component. Frames add a simple border, *shadow frames* add a drop shadow, and *title frames* add a banner.

glue, hglue, vglue Interactors that act as spacers between components of a scene.

graphic Base class for structured graphics objects.

graphic block An interactor that displays a structured graphics object.

immediate-mode graphics A graphics model in which individual geometric shapes are drawn by routines that simply modify pixels on the screen as they are called.

interactor The base class for interactive objects such as menus and buttons.

message An interactor that displays a string of characters.

mover An interactor that scrolls another interactor by some increment.

painter An object providing immediate-mode graphics operations and operations for setting graphics state parameters.

panner An interactor that supports continuous two-dimensional scrolling and incremental scrolling and zooming.

perspective A subject that maintains scrolling and zooming information, including the total size of a view and how much is currently visible.

phrase A clause that places its components end-to-end on a single line.

picture The base class for structured graphics composition objects.

rectangle A graphic that represents and draws a rectangle.

scene The base class for objects that compose interactors; *monoscenes* are scenes that contain only one component.

sentence A clause that places as many of its components as possible on the same line and begins a new line if necessary.

slider A two-dimensional scroll bar.

structured graphics A graphics model that supports hierarchical composition of graphical elements; support is usually provided for coordinate transformations, hit detection, and automatic screen update.

structured text A graphics model that allows hierarchical composition of textual elements, emphasizing the arrangement of elements to make use of available space.

subject An object that maintains state and operations that underlie a user interface; a subject maintains a list of views to be notified when the subject's state changes.

text The base class for structured text objects.

text block An interactor that displays a structured text object.

text list A clause that arranges its components either horizontally or vertically depending on available space.

tray A scene that maintains constraints on the placement of potentially overlapping components.

view An object that provides the user interface to a subject.

viewport A monoscene that can scroll and zoom its component.

whitespace A text object used to introduce space between other text objects in a clause.

word A text object that represents and draws a string of characters.

zoomer An interactor that magnifies or reduces another interactor.

tiles them vertically. Glue between interactors in a box provides space between components. We use *hglue* in *hboxes* and *vglue* in *vboxes*.

Each interactor defines a preferred or *natural* size and the amount it can stretch

or shrink to fill available space. We can use glue of various natural sizes, shrinkabilities, and stretchabilities to describe a wide variety of interface layouts and resize behaviors.

Figure 4 depicts schematically how the

elements of the dialog box are composed using boxes and glue. The corresponding object structure is shown in Figure 5, and the C++ code that implements the dialog box appears in Figure 6. The message and button interactors are each placed in an

hbox with hglue on either side of them. The hglue to the left of the message has a natural size of $\frac{1}{4}$ -inch and cannot stretch, while the glue on the right has a natural size of zero and can stretch infinitely (as specified by the constant *hfil*). If the dialog box is resized as in Figure 7, the margin to the left of the message will not exceed $\frac{1}{4}$ -inch, while the space to the right can grow arbitrarily. Similarly, the button has infinitely stretchable hglue to its left and fixed-size hglue to its right, so that the margin to the right of the button will not exceed $\frac{1}{4}$ -inch.

The hboxes are composed vertically within a vbox, separated by pieces of vglue. The pieces of vglue above the message and below the button have a natural size of $\frac{1}{4}$ -inch, while the vglue between the message and the button has a natural size of $\frac{1}{2}$ -inch. The inner vglue can stretch twice as much as the outer two pieces of vglue. When resized, therefore, the message and button interactors will remain twice as far apart from each other as they are from the edge of the dialog box.

Tray. Suppose we want a dialog box centered atop another interactor, perhaps to notify the user of an error condition. Furthermore, we want the dialog box to remain centered if the interactor is resized or repositioned. Boxes and glue are inappropriate for this type of nontiled composition.

The *tray* scene subclass provides a natural way to describe layouts in which components “float” in front of a background. A tray typically contains a background interactor and several other components whose positions are determined by a set of alignments. For example, the background interactor might display the text in a document; other components could include various messages, buttons, and menus.

Each alignment of a tray component is to some other target interactor, which can be another component of the tray or the tray itself. The alignment specifies a point on the target, a point on the component, and the characteristics of the glue that connects the alignment points. An alignment point can be a corner of the interactor, the midpoint of a side, or the center. The tray will arrange the components to satisfy all alignments as far as possible. If necessary, the components and the connecting glue will stretch or shrink to satisfy the alignments.

Figure 8 shows a simple application in which a tray composes a textual interface and a dialog box. The interactor contain-

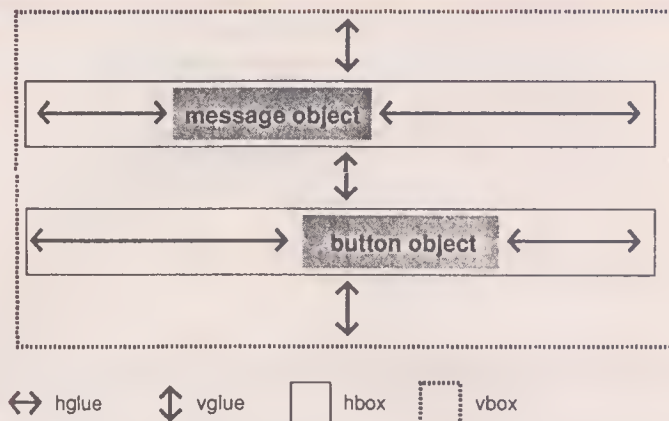


Figure 4. Schematic of dialog box composition using boxes and glue.

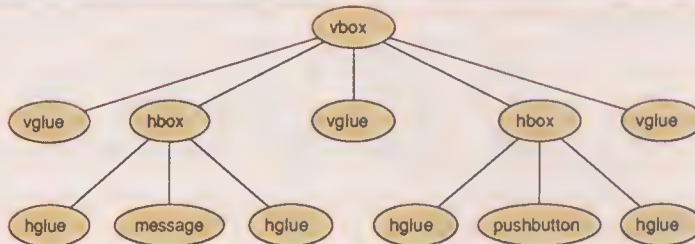


Figure 5. Object structure of dialog box composition.

```
const int space = round(.25*inches);
ButtonState* status;

Frame* frame = new Frame(
    new VBox(
        new VGlue(space, vfil),      /* (natural size, stretchability) */
        new HBox(
            new HGlue(space, 0),
            new Message("hello world"),
            new HGlue(0, hfil)
        ),
        new VGlue(2*space, 2*vfil),
        new HBox(
            new HGlue(0, hfil),
            new PushButton("goodbye world", status, false),
            new HGlue(space, 0)
        ),
        new VGlue(space, vfil)
    )
);
```

Figure 6. C++ code for composing the dialog box interface.

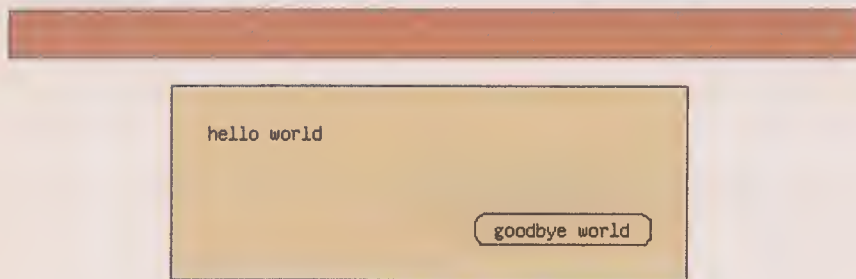


Figure 7. The dialog box after resizing.

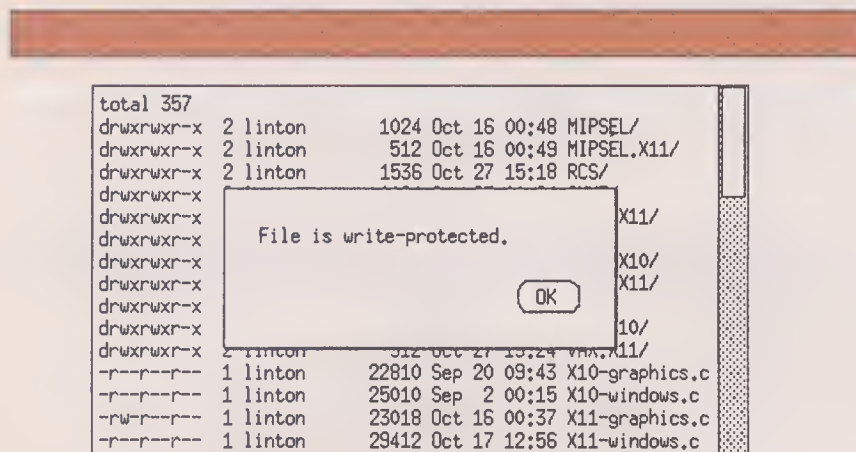


Figure 8. An interface using a tray.

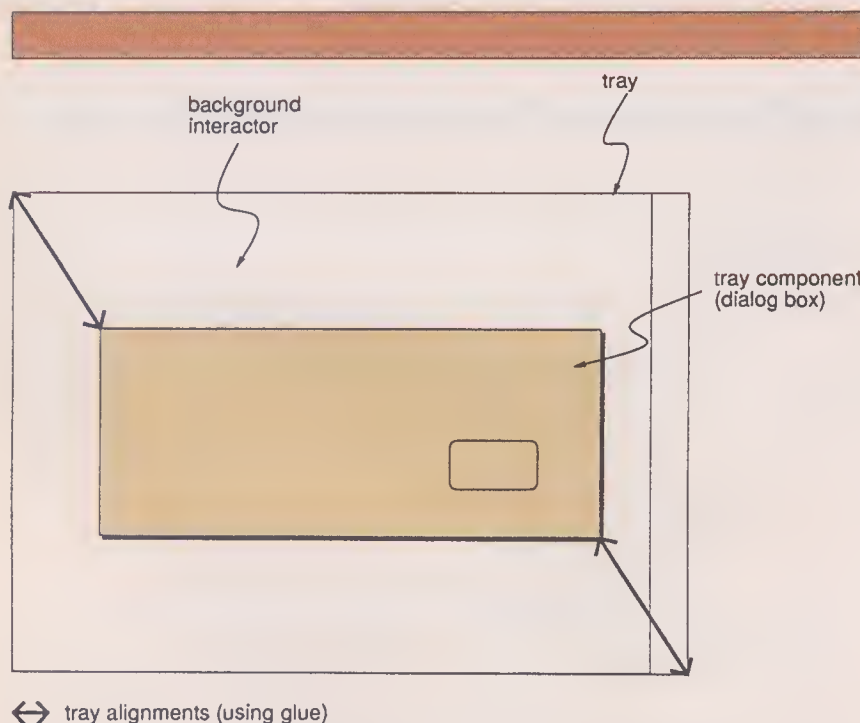


Figure 9. Schematic of tray interface.

ing text and a scroll bar are composed with an hbox and placed into the tray as its background. When the dialog box is required, it is inserted into the tray with its upper left and lower right corners aligned to the corresponding corners of the tray. Figure 9 shows the arrangement of components, and Figure 10 gives the code that implements the interface. The alignments interpose stretchable but nonshrinkable glue with a natural size of $\frac{1}{2}$ -inch to maintain a minimum spacing between the edges of the tray and the dialog box. These alignments guarantee that the dialog box will remain centered atop the background interactor after resizing (see Figure 11). Note how the tray shrank the dialog box to satisfy the alignment constraints once the glue reached its minimum size.

Deck. Another common interface allows the user to flip (rather than scroll) through “pages” of text or graphics as through a book. We can build such an interface in InterViews by composing interactors with a *deck*. The interactors in a deck are conceptually stacked on top of each other so that only the topmost interactor is visible (see Figure 12). The deck’s natural size is determined by the natural size of its largest component. A set of operations allow “shuffling” the deck to bring the desired component to the top.

Decks can be used in other contexts as well. A set of color or pattern options in a dialog box could be composed with a deck, allowing the user to flip through them until reaching the desired choice. Alternate menu entries could be stored in a deck and inserted into a menu to allow changes in the menu’s appearance without rebuilding it each time.

Single component scenes. Boxes, trays, and decks have arbitrary numbers of components. InterViews also provides several scenes that can have only one component. Such scenes are derived from the *monoscene* scene subclass and serve two purposes.

Some monoscenes serve as containers that surround another interactor. The frame used to place a border around the dialog box in the subsection “Boxes and glue” is one example. Other examples include shadow frame, which adds a drop shadow to its component, and title frame, which adds a banner. A *viewport* is a monoscene that scrolls an interactor larger than the available space. Viewports are useful for providing a scrolling interface to nonscrolling interactors.

Graphic composition

The programmer of such systems must provide underlying representations for the graphical objects and define the operations they perform. InterViews provides a collection of structured graphics objects that simplifies the programmer's task.

A simple drawing editor. Figure 13 depicts a drawing editor application in which the user can draw, move, and rotate rectangles and scroll and zoom the draw-

```
const int space = round(.125*inches);
TGlue*g1 = new TGlue(space, space, 0, hfil, 0, vfil);
TGlue*g2 = new TGlue(space, space, 0, hfil, 0, vfil);
/*(width, height, hshrink, hstretch, vshrink, vstretch) */

Tray*tray = new Tray(
    new HBox(
        view,
        new VBorder(1),
        new VScroller(view)
    )
);

tray->Insert(dialog);
tray->Align(TopLeft, dialog, g1);
tray->Align(BottomRight, dialog, g2);
```

Figure 10. C++ code for composing the tray interface.

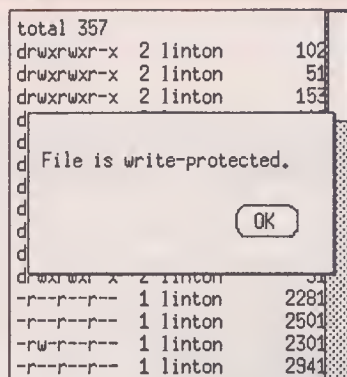


Figure 11. Tray interface after resizing.

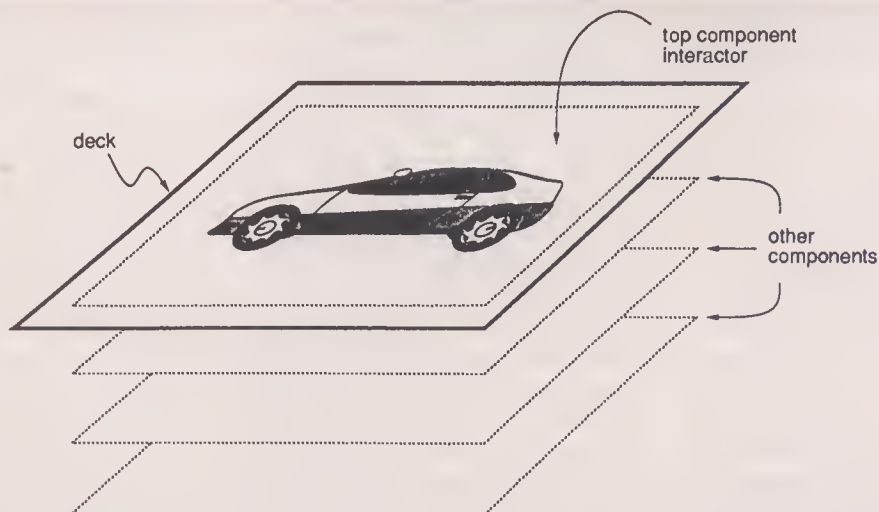


Figure 12. Composition using a deck.

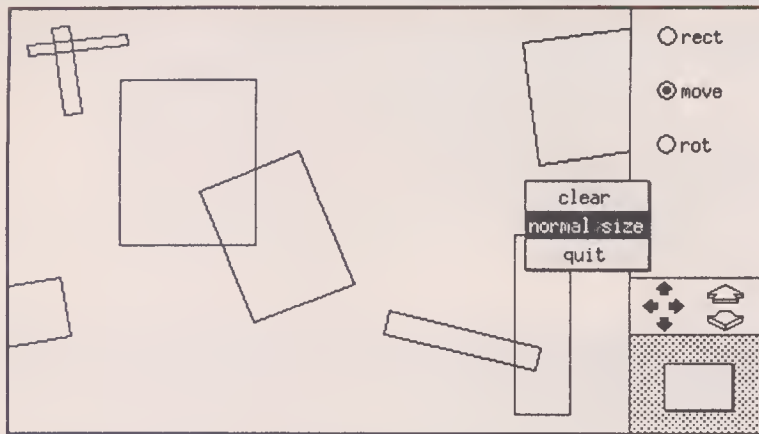


Figure 13. A simple drawing editor application.

ing area. To draw a rectangle, the user presses the "rect" button and drags out a rectangle in the drawing area. An existing rectangle can be moved or rotated by pressing the appropriate button and dragging the rectangle.

In each of these operations, the drawing editor provides animated feedback as the user creates and manipulates rectangles. Animation reinforces the user's belief that he or she is manipulating real objects. As a rectangle is moved, for instance, its outline follows the mouse; during rotation, the outline revolves about the rectangle's center. Such dynamic feedback is characteristic of a direct manipulation editor.

Implementing the drawing editor. A programmer can compose the elements of the user interface using InterViews interactor and graphic subclasses as shown in Figure 14. The buttons are instances of *radio button*, a predefined subclass of the button class. The interface to scrolling and zooming is provided by a *panner*, the two-dimensional scroller in the lower right of the interface. The drawing area in which the rectangles appear is a *graphic block*, an interactor that displays structured graphics objects. These elements are composed using boxes and glue. The editor's pop-up command menu, appearing in the center-right of Figure 13, is an instance of the

menu class.

Each rectangle in the drawing is an instance of the rectangle class, a subclass of graphic. The rectangles are composed in a picture, and the picture is placed in the graphic block. The graphic block translates and scales the picture to implement scrolling and zooming. Rectangles are moved and rotated by calling transformation operations on the rectangle objects. The picture performs hit detection by returning the component that corresponds to a coordinate pair.

Semantics of graphic composition. The drawing editor demonstrates simple composition of graphics. In this example, the hierarchy of graphical objects is only one level deep; all the rectangles are children of a single parent picture. Of course, more complex hierarchies are common in a practical drawing editor. However, even the simple one-level hierarchy demonstrates the semantics of graphic composition. For example, when the graphic block applies a transformation to the picture to scroll or zoom it, the transformation affects all the rectangles in the picture. Furthermore, altering any of the picture's graphics state attributes affects its children as well. For example, changing the picture's brush width attribute also changes the brush widths of its children.

The composition mechanism defines how the picture's graphics state information affects its components. A picture draws itself by drawing each component recursively with a graphics state formed by concatenating the component's state with its own. The default semantics for concatenation are that the attributes defined by a graphic's parent override the graphic's own attributes. If a parent does not define a particular attribute, then the child graphic's attribute is used. Coordinate transformations are concatenated so that the child's transformation precedes the parent's.

These semantics represent a kind of reverse inheritance of graphics attributes, since parents can override their children. This mechanism is useful in editors where operations performed on interior nodes of the graphic hierarchy affect the leaf graphics uniformly. Classes derived from the graphic class can redefine the semantics of concatenation if the default semantics are inappropriate.

Immediate-mode graphics. We normally do not use structured graphics objects to draw scroll bars, menus, or other user interface components that are simple to draw procedurally. Interactors use *painter* objects for this purpose. Painters provide immediate-mode drawing operations (including operations for drawing lines, filled and open shapes, and text) and operations for setting the current fill pattern, font, and other graphics state. The results of a painter drawing operation appear on the display immediately after the operation is performed. The difference between painter-generated graphics and structured graphics is that painters do not maintain state or structure that reflects what has been drawn, so there is no way to access and manipulate the graphics. In contrast, structured graphics objects maintain geometric and graphical state and can be manipulated before and after they are drawn.

Structured graphics is most appropriate where an indefinite number and variety of graphical objects are manipulated directly. It is a powerful tool for constructing graphics editors that provide an object-oriented editing metaphor because structured graphics objects embody the same metaphor. These objects typically represent the data managed by the editor. Painters should be used to draw simple, unchanging elements of the interface that do not justify the storage overhead of graphics objects.

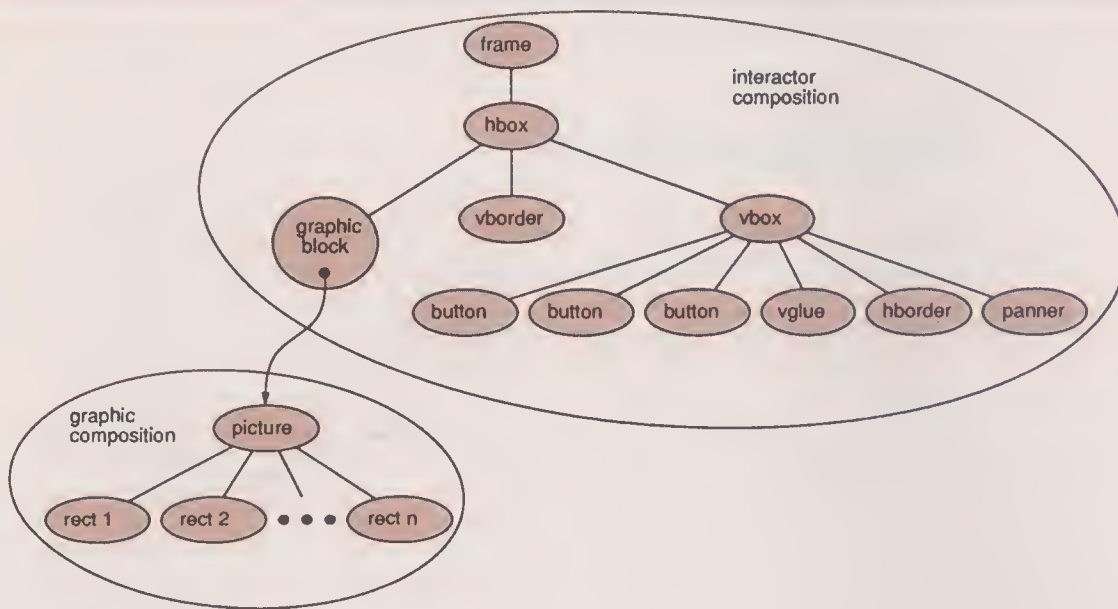


Figure 14. Drawing editor object structure.

Text composition

Direct-manipulation textual interfaces require special support to handle problems in the presentation of text, such as line and page breaking and arranging text to reflect the logical structure of a document. InterViews structured text objects simplify the implementation of direct-manipulation textual interfaces.

A simple class browser. Figure 15 shows the interface to a class browser, a simple application for perusing C++ class declarations. The browser displays a class declaration with the class name underlined and member functions in bold. Clicking on the class name opens a window showing documentation for the class, and clicking on a member function opens a window showing the function's definition. Text composition objects maintain the arrangement of the text. As Figure 16 shows, resizing the window reformats the text to use available space.

Implementing the class browser. Text and clause subclasses compose the text displayed in the browser. Objects of the *word* (a string of characters) and *whitespace* (blank space of a given size) classes are assembled using various composition objects so that the lines of code will fill

```

/* Base class for interactive objects. */
class Interactor {
public:
    Interactor(
        Sensor* in = stdsensor, Painter* out = stdpaint
    );
    ~Interactor();
    void Listen(Sensor*);
    void Iconify();
    void Read(Event&);
    virtual void Resize();
    virtual void Draw();
    virtual void Redraw(
        Coord left, Coord bottom, Coord right, Coord top
    );
    virtual void Handle(Event&);
    ...
};

```

Figure 15. A simple class browser application.

available space in an appropriate manner. The entire composition is placed in a *text block* (an interactor that displays structured text objects), and the text block is inserted into a frame.

Semantics of text composition. Subclasses of clause specify the way their components will be arranged. Different clauses use different strategies for using available space:

```

/* Base class for interactive objects. */
class Interactor {
public:
    Interactor(
        Sensor* in = stdsensor,
        Painter* out = stdpaint
    );
    ~Interactor();
    void Listen(Sensor*);
    void Iconify();
    void Read(Event&);
    virtual void Resize();
    virtual void Draw();
    virtual void Redraw(
        Coord left,
        Coord bottom,
        Coord right,
        Coord top
    );
    virtual void Handle(Event&);
    ...
};

```

Figure 16. The class browser after resizing.

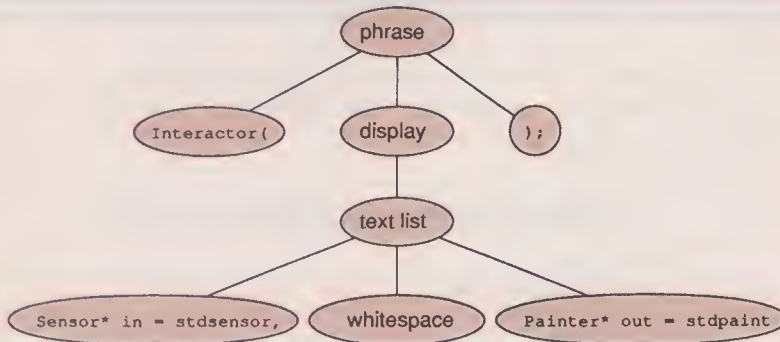


Figure 17. Object structure of the text composition for the Interactor constructor.

```

Interactor(Sensor*in = stdsensor, Painter*out = stdpaint);

Interactor(
    Sensor*in = stdsensor, Painter*out = stdpaint
);

Interactor(
    Sensor*in = stdsensor,
    Painter*out = stdpaint
);

```

Figure 18. Possible layouts of the Interactor constructor.

- A *phrase* formats its components without regard to space. The components are simply placed end-to-end on a single line.

- A *text list* can arrange its components either horizontally or vertically. If the whole list will not fit in a horizontal format, then the list will place each component on a separate line. Text lists are used in the browser for composing the member function parameter lists.

- A *display* defines an indented layout. If the display will not fit on the current line, then it is placed on the following line with a specified indentation. The browser composes class and member function declarations using displays.

- A *sentence* will place as many components as possible on the current line and will begin a new line if necessary. The browser uses sentences for comments.

To illustrate how we can use text composition, consider the composition of the Interactor constructor in the browser (see Figure 17). The declaration is composed as a phrase with three components: the first component is a word representing the string Interactor(, the second is a display that contains a text list of the formal parameters, and the third is a word representing the string); .

Figure 18 shows that the constructor declaration will appear in one of several layouts depending on the available space. In the top example, all the text can fit on a single line. In the middle example, the available space has been reduced so that there is not enough room for the display containing the parameter list; the display is placed on a separate, indented line. In the bottom example, the available space has been reduced further, causing the text list to display vertically instead of horizontally.

Text composition is most useful when the interface requires direct manipulation of text, when the text should reflect the structural characteristics of the document, or when the text layout should automatically make good use of available space. Painters are more appropriate for embellishing interfaces with simple, noninteractive text.

Subjects and views

In InterViews we distinguish between interactive objects, which implement a user interface, and abstract objects, which encapsulate the underlying data. We refer to interactive and abstract objects as *views* and *subjects*, respectively. This separation

Making user interface development easier

We can divide software systems that facilitate construction of graphical user interfaces into two broad categories: toolkits and user-interface management systems.

Toolkits. A user interface toolkit provides programming abstractions for building user interfaces. InterViews, the X Toolkit, and the Andrew Toolkit¹ are good examples. The X Toolkit defines widget and composite classes analogous to interactors and scenes in InterViews. Tiling composites include *box* and *vpaned*, and the *form* composite allows its components to overlap. Composite objects maintain a pointer to a *geometry manager* function that is responsible for the proper layout of components. The geometry manager can be replaced at runtime to change the layout strategy.

The Andrew Toolkit includes objects that comprise the data to be edited, such as text, bitmaps, and more sophisticated objects such as spreadsheets and animation editors. Its composition mechanism allows these objects to be embedded in multimedia documents.

In addition to standard toolkit functionality, Graphical Object Workbench² allows the programmer to specify constraints between objects. Constraints can enforce dependencies between individual pieces of data. For example, the programmer can specify that a value stored in one object is a function of a value in another object. Grow also has graphical constraints for confining and connecting graphical objects. Such constraints can guarantee that a graphical object stays within a prescribed area or that two visually connected objects stay connected when one or the other is translated.

Smalltalk Model-View-Controller³ and its descendant, Apple's MacApp,⁴ are among the earliest and best-known object-oriented toolkits. MacApp differs from newer toolkits in that it implements the particular "look and feel" of Macintosh applications. MVC is unique in that it divides interface components into *model*, *view*, and *controller*. Models are similar to subjects in InterViews, controllers are responsible for input handling, and views are responsible solely for output. In contrast, other toolkits that distinguish between interactive and abstract objects put the functionality of MVC controllers and views into a single object (corresponding to an InterViews view) that handles input and output. This consolidation reflects the tight coupling between input and output in direct-manipulation interfaces. Placing responsibility for input and output in the same object reduces the total number of objects and the communication overhead between them, simplifying the toolkit and potentially increasing its efficiency.

UIMSS. User-interface management systems are generally

characterized by

- (1) complete separation of the code that implements the user interface to an application from the code for the application itself, and
- (2) support for specifying the user interface at a higher level of abstraction than general-purpose programming languages.

UIMSSs separate interface and application for some of the same reasons that many toolkits separate subjects and views, namely to isolate application code and interface specification and to allow different interfaces to the same application. However, UIMSSs do not implement any application code, whereas subjects usually do. Moreover, UIMSSs minimize the interaction between the application and the interface to maximize their independence. UIMSSs generally concentrate on abstracting the syntax and semantics of the user interface. Their main goal is to let interface designers and even end users design and modify the interface quickly without requiring extensive programming skills or knowledge of the application. To avoid conventional programming, UIMSSs use special-purpose languages or other formalisms such as finite-state transition diagrams to describe the appearance of the interface and the kinds of interaction it supports. In most UIMSSs the specification is interpreted by a runtime system incorporated into the application.

A widely known and used UIMS is Apollo's Domain/Dialog.⁵ The package consists of a compiler and a runtime library. The compiler reads a declarative description of the user interface and how it connects to the underlying application. It then generates a more compact description that is interpreted by the runtime library.

The user interface is specified in terms of interaction techniques, which correspond to primitive interface components, and structuring techniques, which are the composition mechanisms for the primitives. Domain/Dialog defines structuring techniques for arranging components into rows and columns and a "oneof" technique that displays only a single component. These structuring techniques allocate space for their components in a manner similar to InterViews' boxes and glue; they request a minimum, maximum, and optimal size from their components and distribute the available space among them.

Domain/Dialog places greater emphasis on composition than most UIMSSs, which center more on how to specify the input and output behavior of a user interface without conventional programming. Sassafras,⁶ a prototype UIMS developed at the University of Toronto, focuses on supporting concurrent user input from multiple devices and on efficient communica-

is important in many aspects of user interface design. It is a vehicle for customization, allowing programmers to present different, independently customizable interfaces to the same data. It is a useful structuring mechanism that separates user interface code from application code. It permits different representations of the

same data to be displayed simultaneously such that data changes made through one representation are immediately reflected in the others. Several other user interface packages support this separation, including the Andrew Toolkit, Smalltalk Model-View-Controller, Graphical Object Workbench, and MacApp (see the sidebar enti-

tled "Making user interface development easier").

Views in InterViews are typically implemented with compositions of interactors, graphics, and text objects. Subjects are often (but need not be) derived from the subject class. A subject maintains a list of its views. Views define an Update opera-

tion and synchronization between the modules that support user interaction. Syngraph⁷ takes a description of a user interface written in a formal grammar and generates Pascal code that implements it. Recent work by Foley et al.⁸ uses a knowledge base describing the interface to raise the level of abstraction beyond detailed assembly of components.

Another class of UIMS lets designers create a user interface by direct manipulation instead of textual specification. Research systems such as Cardelli's dialog editor⁹ and Myers' Peridot¹⁰ and commercial systems such as SmethersBarnes' Prototyper¹¹ let designers draw the user interface using a drawing editor-like metaphor. The system then generates routines that must be incorporated into the application. Cardelli's system lets designers specify the resize semantics using attachment points; an edge of a component can be attached to an arbitrary point in the interface. The component will stretch or shrink if necessary to maintain the attachment. Peridot allows the designer to specify many aspects of the interface by demonstration, inferring the proper semantics of the interface from the designer's actions. Prototyper provides a drawing editor interface to building Macintosh applications and is one of the few commercial direct-manipulation interface editors.

Toolkits, UIMSs, and InterViews. There is a growing interest in toolkits, while many researchers have begun to question whether UIMSs really help.¹² Early non-object-oriented toolkits^{13,14} were criticized as being too low-level and difficult to use, thus widening interest in UIMSs. Yet few UIMSs have gained wide acceptance. Researchers¹⁵ have identified several shortcomings of existing UIMSs:

Limited range of interfaces. Since UIMSs allow interface specification at a high level, they necessarily limit the range of interfaces they can create. This is especially true of direct-manipulation interface editors, which must rely on graphical or demonstrational specification of the interface's semantics.

Reliance on an interpreted specification language. The special-purpose language used by a traditional UIMS is likely to be unfamiliar to programmers and interface designers alike. Moreover, the language is usually inferior to established general-purpose languages, the debugging tools are primitive or nonexistent, and runtime overhead associated with interpreting the specification often degrades performance compared to conventional implementations.

Inadequacy for direct manipulation interfaces. The strict separation of application and interface code usually results in a low-bandwidth connection between the two. Thus, most UIMSs do not support interfaces requiring real-time response

to user input, such as those using rubberbanding or other animated effects.

Difficulty in adapting to change. The time needed to produce UIMSs makes it difficult to keep them in step with the latest interface designs. The problem only gets worse as interfaces become more complex.

Because InterViews is a toolkit, it avoids the problems associated with UIMSs. InterViews is distinguished from other toolkits in its variety of composition mechanisms (tiled, overlapped, stacked, constrained, and encapsulated), its support for nonlinear deformation (independent stretching and shrinking) of interactors, and its object-oriented approach to structured graphics and text. InterViews simplifies the creation of both the controlling elements of the interface (buttons and menus) and the data to be manipulated (text and graphics objects). InterViews thus offers comprehensive support for building user interfaces.

References

1. A.J. Palay et al., "The Andrew Toolkit: An Overview," *Proc. 1988 Winter USENIX*, Feb. 1988, pp. 9-21.
2. P.S. Barth, "An Object-Oriented Approach to Graphical Interfaces," *ACM Trans. Graphics*, Vol. 5, No. 2, Apr. 1986, pp. 142-172.
3. G.E. Krasner and S.T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *J. Object-Oriented Programming*, Vol. 1, No. 3, Aug.-Sept. 1988, pp. 26-49.
4. Apple Programmer's and Developer's Assoc., *MacApp: The Expandable Macintosh Application*, Renton, Wash., 1987.
5. A. Schuler et al., "ADM — A Dialog Manager," *Proc. ACM CHI 85*, Apr. 1985, pp. 177-183.
6. R.D. Hill, "Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction — The Sassafras UIMS," *ACM Trans. Graphics*, Vol. 5, No. 3, July 1986, pp. 179-210.
7. D.R. Olsen and E.P. Dempsey, "Syngraph: A Graphical User Interface Generator," *ACM SIGGraph 83*, July 1983, pp. 43-50.
8. J. Foley et al., "A Knowledge-Based User Interface Management System," *Proc. ACM CHI 88*, May 1988, pp. 67-72.
9. L. Cardelli, "Building User Interfaces by Direct Manipulation," Tech. Report 22, Digital Equipment Corp. Systems Research Center, Oct. 1987.
10. B.A. Myers, *Creating User Interfaces by Demonstration*, PhD thesis, Tech. Report CSRI-196, Computer Science Dept., Univ. of Toronto, 1987.
11. SmethersBarnes, *Prototyper Manual*, Portland, Ore., 1987.
12. J. Rosenberg et al., "UIMSs: Threat or Menace?," *Proc. ACM CHI*, May 1988, pp. 197-200.
13. Apple Computer, *Inside Macintosh, Volume I*, Addison-Wesley, Reading, Mass., 1985.
14. Sun Microsystems, *Sun Windows Programmers' Guide*, Mountain View, Calif., 1984.
15. B.A. Myers, "Tools for Creating User Interfaces: An Introduction and Survey," Tech. Report CMU-CS-88-107, Carnegie Mellon Univ., Jan. 1988.

tion responsible for reconciling the view's appearance with the current state of the subject. Calling Notify on a subject in turn calls Update on its views, thus enabling the views to update their appearance in response to a change in the subject.

In practice it is inconvenient to force every user interface concept into the sub-

ject/view model. For example, it is unnecessary to associate a subject with every menu because interfaces seldom require multiple views of the same menu. However, many InterViews library components do use the subjects and views paradigm. Two examples relate to the implementation of scrolling and buttons.

Scrolling and perspectives. An interactor that supports scrolling and zooming maintains a *perspective*. The perspective is a subject that defines a range of coordinates representing the total extent of the interactor's output space and a subrange for the currently visible portion of the total range. For example, in the drawing editor

mentioned above, the total extent of the graphic block's perspective is obtained from the picture's bounding box; its subrange is the space the graphic block occupies on the screen. In a text editor the vertical range might be the total number of lines in a file; the subrange would be the number of lines displayed by the editor on the screen.

Scrolling and zooming are performed by modifying the interactor's perspective. An interactor can modify its own perspective (when the text editor adds a line to the file, for example), or the perspective can be modified by the user manipulating one of its views.

The panner in the drawing editor is a view of the perspective associated with the editor's graphic block. The panner is really a composition of several other perspective views: a *slider*, a set of four *movers*, and two *zoomers*. Each of these elements views the same perspective; the slider scrolls the drawing along both the *x* and *y* axes, each mover provides incremental scrolling in one of four directions, and the zoomers respectively enlarge and reduce the drawing. The number of views on the same perspective is unlimited; a change made through one view of a perspective will be reflected in all its views.

The advantage of this organization is that one view of a perspective need not know about other views of the same perspective. Whenever the perspective is changed, either by the interactor or by a view, all the views are notified. Each view of the perspective is responsible for updating its appearance appropriately in response to the change. For example, when a mover or zoomer is pressed, the perspective is updated and the slider is notified automatically. The slider can then redraw itself to reflect the new perspective.

Figure 19 shows how a graphic block's perspective coordinates the scrolling operation when the user presses one of the panner's movers. The graphic block modifies its perspective on behalf of the mover because the graphic block might want to limit the amount of scrolling. In this instance, the perspective and the interactor are considered together as the subject to which views such as panners are attached.

Buttons and button states. The example dialog box uses a button for dismissal. In InterViews, a button is a view of a *button state* subject. When the user presses a button, the button sets its button state to a particular value. Several buttons can view

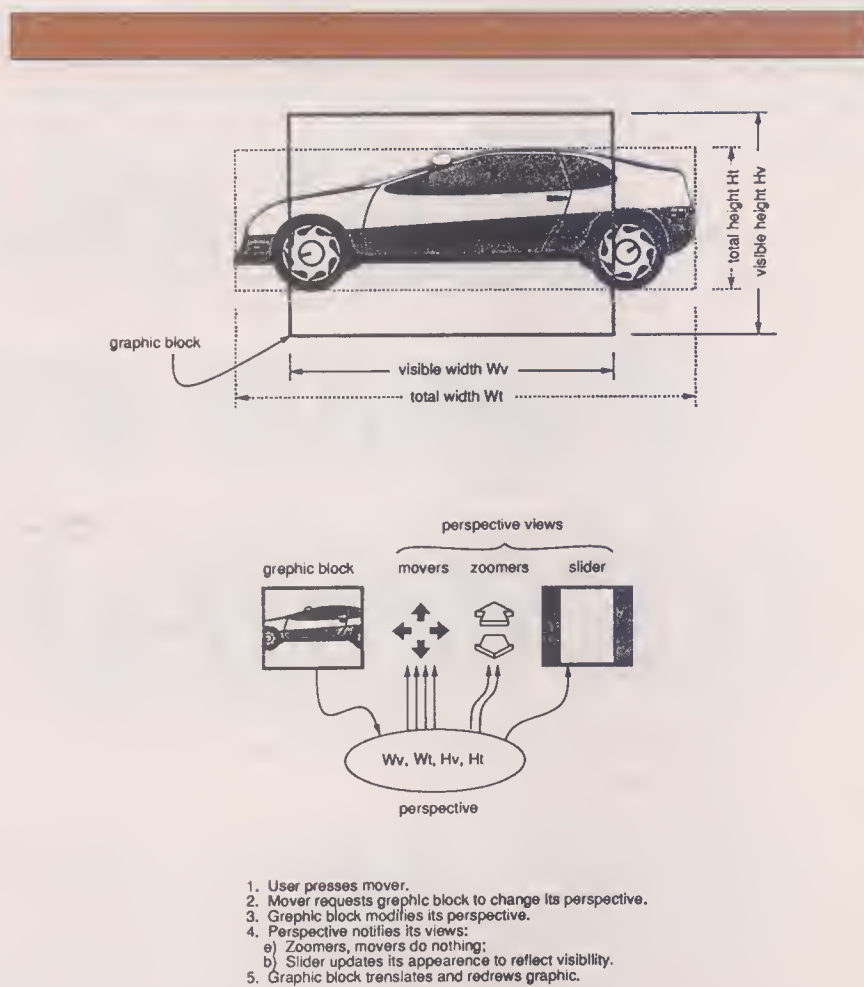


Figure 19. How a perspective coordinates scrolling of a graphic block.

a single button state; like any subject, a button state notifies all its views (buttons) when it changes.

To illustrate this, consider how InterViews radio buttons are implemented. A radio button acts like a tuning button on a car radio; only one button in a group of radio buttons can be "on" at a time. Radio buttons are provided when the user should select an option from several mutually exclusive choices. A single button state is the subject for a group of radio buttons. Pressing one of the radio buttons sets the button state to a particular value. The button stays pressed until the button state is changed to a different value, usually by pressing another radio button in the group.

Customization

InterViews adopts the X Toolkit⁴ model to support customization of interactors. Users can define a hierarchy of attribute names and values. An interactor can retrieve the value of an attribute by name; it interprets the value to customize some aspect of its appearance or behavior. Attribute lookup involves a search through parts of the attribute hierarchy that match the interactor's position in the object instance hierarchy. Each interactor can have an instance name; interactors not explicitly named inherit a class name. The name given the interactor at the root of the instance hierarchy is usually the name of the application.

For example, suppose the application containing the example dialog box was called "hello," and the push button in the dialog box had the instance name "bye." The full name of the attribute that specifies the font for the button label would then be

```
hello.Frame.VBox.HBox.bye.font
```

Attribute names can include wild-card specifications so that one attribute can apply to several interactors. The font of the push button in the example dialog box is more likely to be specified by an attribute named `hello*PushButton.font`, which would apply to any push button in the application, or even `*font`, which would apply to any font in any application. The mechanism for accessing attributes ensures that the attribute with the most specific name is the one used to satisfy a query. The InterViews library automatically handles standard attributes such as "font" and "color."

The designer of an application chooses names for interactors that users can customize. Users specify these names to refer to interactors they want to customize. Consistency across a range of applications is achieved by a consistent choice of instance and attribute names. For example, all confirmation buttons in all "quit" dialog boxes will be red if the user lists the attribute `*quit*OK.background:red`, if all quit dialog boxes are given the instance name "quit," and if all confirmation buttons are named "OK."



Mark A. Linton is an assistant professor in the Computer Systems Laboratory of the Electrical Engineering Dept. at Stanford University. His current research interests are in programming environments, user interfaces, operating systems, and workstation architectures.

Linton received the BSE degree from Princeton University in 1978 and the MS and PhD degrees in computer science from the University of California at Berkeley in 1981 and 1983, respectively. He is the author of the Unix debugger "dbx" and the Stanford workstation benchmarks.

Current status

InterViews currently runs on MicroVAX, Sun, Hewlett-Packard, and Apollo workstations on top of the X Window System⁵ versions 10 and 11. The library comprises roughly 30,000 lines of C++ source code, of which about 2,000 lines are X-dependent. InterViews applications do not call X routines directly and are thus isolated from the underlying window system.

We have implemented several applications on top of the library, including a scalable digital clock, a load monitor, a drawing editor, a reminder service, a window manager, and a display of incoming mail. The applications have been used daily by about 20 researchers for nearly two years, and the library is being used in development efforts at Stanford, at other universities, and in industry. We are currently using InterViews in the development of a more general drawing system, a program editor, a visual command shell, and a visual debugger.

Our experience with InterViews has convinced us of the importance of object-oriented design, subject/view separation, and composition in facilitating the implementation of user interfaces. Composition is particularly important. Providing one or two ways to combine interface elements is not enough. To really help the programmer, a user



John M. Vlissides is a research assistant in the Computer Systems Laboratory at Stanford University, where he is pursuing a PhD in electrical engineering. His dissertation work is on a generalized model for constructing graphical editing systems. His research interests also include programming environments, constraint systems, and simulation.

Vlissides earned a BS degree in electrical engineering from the University of Virginia, where he received the Electrical Engineering Dept.'s Chairperson's prize, in 1983 and an MS degree in electrical engineering from Stanford in 1985. He is a member of Eta Kappa Nu and Tau Beta Pi, and a student member of the IEEE.

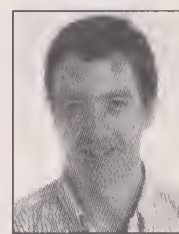
interface toolkit must offer a rich set of composition mechanisms along with a variety of predefined objects. The programmer should be able to pick and choose from among the predefined components for the bulk of the interface, and the toolkit should make it easy to synthesize components unique to the application. The composition mechanisms in InterViews make this possible. □

Acknowledgments

Several people have contributed to the design and implementation of InterViews. Craig Dunwoody and Paul Hegarty participated in the design of the basic protocols. Paul also developed the window manager application, and John Terrante implemented the drawing editor. We are grateful to the growing InterViews user community for its encouragement and support. This work was funded by the Quantum project through a gift from Digital Equipment Corporation.

References

1. K. Potosnak, "Do Icons Make User Interfaces Easier to Use?", *IEEE Software*, Vol. 5, No. 3, May 1988, pp.97-99.
2. B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, Reading, Mass., 1986.
3. D. Knuth, *The TeXbook*, Addison-Wesley, Reading, Mass., 1984.
4. J. McCormack, P. Asente, and R.R. Swick, *X Toolkit Intrinsics-C Language Interface*, Digital Equipment Corp., Mar. 1988.
5. R.W. Scheifler and J. Gettys, "The X Window System," *ACM Trans. Graphics*, Vol. 5, No. 2, Apr. 1986, pp. 79-109.



Paul R. Calder is a research assistant in the Computer Systems Laboratory at Stanford University, where he is pursuing a PhD in electrical engineering. His dissertation work is on structured program editing. His research interests also include programming environments and user interfaces.

Calder received a BE(Elec) in 1978 and a MEngSc(Biomed) in 1980, both from Melbourne University, Australia. From 1980 to 1987 he served on the faculty of the Electrical Engineering Dept. at Ballarat College of Advanced Education, Australia. He is a member of the IEEE and the ACM.

Readers may contact Mark Linton at the Center for Integrated Systems, Rm. 213, Stanford University, Stanford, CA 94305.

GAUSS RUN WITH
THE BEST

Saves time. Saves money. New GAUSS 2.0.

Now you can run most routines at or near the capacity of your hardware.

From Bell Labs to CalTech, research scientists and engineers have found GAUSS can help them solve problems with their PCs that would take them hours longer with conventional languages.

And not only does GAUSS run fast, it's easier to program because its language is closer to mathematical syntax. GAUSS allows you to easily enter your own ideas even if you are not a programmer and saves time by including over 250

mathematical routines. You can even call your favorite Fortran, C or assembler subroutines.

Now GAUSS 2.0, the second generation of the new breed of matrix handling programming languages is available, with indexed, easier-to-use documentation, on-line debugger and Smart™ on-line help that can even handle your own functions. And GAUSS 2.0 is faster, up to 20% on some calculations, compared to our previous versions.

The GAUSS system includes publication quality graphics and a library of mathematical

routines and applications, documented in the updated 2.0 manual. All this for the total price of just \$395. So don't waste another minute

running at half speed, order GAUSS 2.0 today.

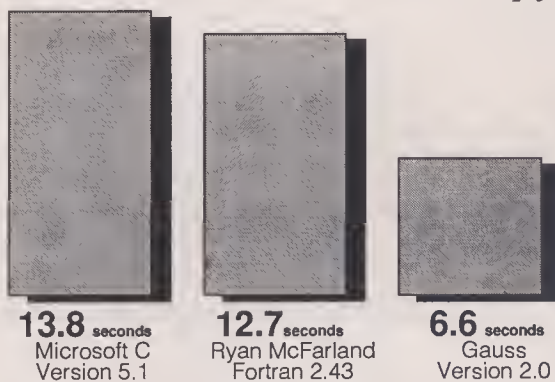
We're so sure GAUSS will make you more productive, giving you more answers to more questions more quickly, that we guarantee it.

If you are not completely satisfied with the power, speed and flexibility of

GAUSS simply return it for a full refund. No questions asked. Call (206) 631-6679 to order.

GAUSS - faster, easier, smarter...guaranteed.

90 x 90 Double Precision Matrix Multiply



GAUSS 2.0

FOR IBM PC-XT-AT-PS/2 & COMPATIBLES
WITH MATH COPROCESSOR

BY APTECH SYSTEMS, INC.

26250 - 196th Place SE

Kent Washington 98042

Phone (206) 631-6679

FAX (206) 630-1220

CALL OR WRITE FOR FREE BROCHURE

IBM IS A REGISTERED TRADEMARK OF INTERNATIONAL BUSINESS MACHINES
ALL TESTS RUN ON AN IBM PS/2 MODEL 80 16 MHZ.

MULTIPLY THE BENEFITS OF CASE

with the LAN BASED VISIBLE SOLUTION®

Coherent project development builds high quality systems faster!!!

All project team members use the same data dictionary!!!

Methodology based expert rules checks all work for consistency across the entire project!!!

Flexible, friendly, easy-to-use graphical interface brings project staff up-to-speed quickly!!!

VISIBLE ANALYST® WORKBENCH

THE NETWORKABLE VERSION 3.0--NEW FEATURES

Identical look & feel as single user version • Configurable executable modules • Unlimited number of potential users on the LAN • Little or no system degradation resulting from Workbench use • Project wide rules validation • Multiple concurrent users • Common data dictionary • Full file and record locking • Multiple security levels •

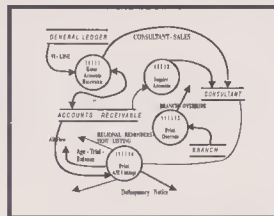
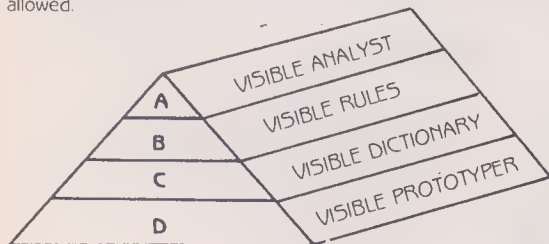
Ever since computer aided software engineering products came on the market, interest has been high. And so, not surprisingly, has been the cost of owning them. But now Visible Systems is offering state of the art CASE technology at an incredibly affordable price. \$595.00 per module. These modular tools give your company the ability to grow in CASE power. The modules of the Workbench build upon each other to provide more and more productivity.

VISIBLE ANALYST - MODULE A

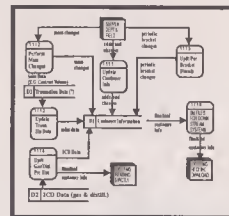
The Visible Analyst is the foundation of the Workbench. The user can create any analysis or design diagram. Multiple line styles, text fonts, and symbol types are standard features. Other features include a custom symbol generator and a reusable constructs library. Build and store entire diagrams for reuse in other projects. Boilerplates allow diagrams to be customized. Diagrams can be output in presentation quality using HP laser printers. Entire subsystems or single diagrams can be moved from one project to another while maintaining nested relationships. The LAN version allows multiple analysts to build or edit diagrams, within the same project, concurrently!!

VISIBLE RULES - MODULE B

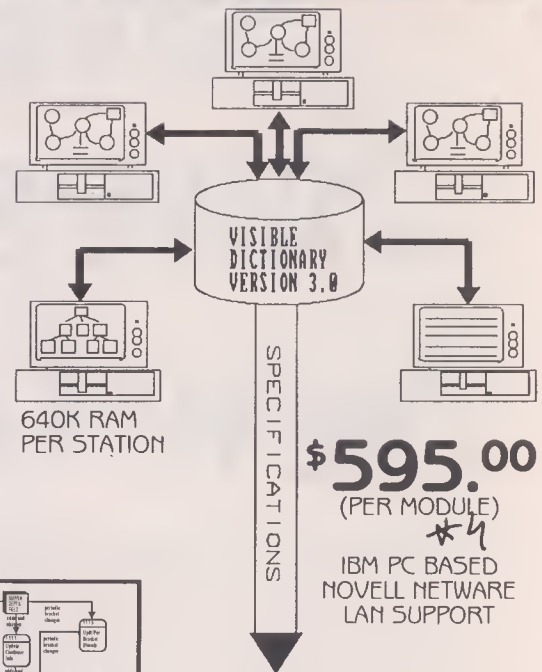
The Visible Rules is the on-line expert system that makes the use of structured techniques possible. The rules module implements either of two powerful, methodology based, expert rules systems: Yourdon/DeMarco or Gane & Sarson. Once enabled, the rules system monitors the diagramming process for naming conventions, data balancing, proper symbol usage etc. After diagrams have been built, the rules module analyzes them for errors. Error reports are output to screen or hardcopy. And the analysis takes place on-line. One never leaves the diagram. Problem resolution and identification is faster and easier. Logical data flow splitting is fully supported. Dictionary is automatically populated. Unlimited levels of nested decompositions are allowed.



YOURDON DIAGRAM



GANE & SARSON



VISIBLE DICTIONARY - MODULE C

Visible Dictionary is the central data repository for the system being modeled. It is multi-user and interactive. Many users may access it simultaneously. Full file and record locking capabilities keep users from colliding and corrupting data. Visible Dictionary has many powerful features including: "where used" listings, wild card searches and finds, full import/export, automatic entry generation and entry updating, on-line accessibility from diagrams, many different entry types, customized report formatting, and much more. The Dictionary organizes, reports, and categorizes all project data. File format information is provided for data transfers in ASCII format. The specification developed using the Visible Workbench becomes a company asset that can be used to build upon in later stages of the systems development lifecycle.

NEW!!! VISIBLE PROTOTYPING - MODULE D

The Visible Prototyper is our new screen design and full system simulation tool. Four levels of prototyping are supported: Screen design and panel linking; Panel branching, Data selection; and System simulation. User interface is "pre-approved". Integrates with all design methodologies, use for education and training -reduces costs. The Prototyper is available as a standalone or integrated fourth module with links to the Visible Dictionary for consistency in data names.

Visible Systems CORPORATION

49 Lexington Street • Newton, MA 02165 • (617) 969-4100

FAX (617) 969-9505 Telex 261102 VSCUR

LEADING EDGE CASE TECHNOLOGY

SAME DAY SHIPMENTS WORLDWIDE

© 1988 Visible Systems Corporation

Visible, Visible Solution, The Visible Analyst and Visible Systems Corporation are registered Trademarks of Visible Systems Corporation

Reader Service Number 4

Performance of Multiprocessor Interconnection Networks

Laxmi N. Bhuyan, University of Southwestern Louisiana

Qing Yang, University of Rhode Island

Dharma P. Agrawal, North Carolina State University

With device characteristics approaching physical limits, parallel or distributed processing has been widely advocated as a promising approach for building high performance computing systems. The continued impetus in research in these areas arises from two factors: (a) the technological development in the area of VLSI chips and (b) the observation that significant exploitable software parallelism is inherent in many scientific and engineering applications.

To exploit this parallelism efficiently, a parallel/distributed system must be designed to considerably reduce the communication overhead between the processors. The communication architecture of the system might support one application well but might prove inefficient for others.

Therefore, we need to take a general approach, independent of the application, while designing the communication system or the interconnection network (IN) of a general-purpose parallel/distributed system. The IN must be efficient, reliable, and cost effective. A complete interconnection, such as a crossbar, might be cost prohibitive, but a shared-bus interconnection might be inefficient and unreliable. Thus, present research is directed to designing INs whose cost and performance lie somewhere between the two extremes.

Multiprocessor designers need analytical techniques to evaluate network performance. This article presents a tutorial on these evaluation tools to guide designers through the design process.

Ongoing research in the area of parallel and distributed processing suggests a number of promising INs. Because of the high cost involved in hardware implementation or software simulation of these INs, performance evaluation of these networks needs to be carried out through analytical

techniques so that we can make a choice between various alternatives. A mathematical model makes it possible to study the efficiency of the IN in terms of various design parameters used as inputs to a model. Therefore, the intent of this article is to provide a tutorial on the subject of performance evaluation of multiprocessor interconnection networks to guide system designers in their design process.

A classification of parallel/distributed systems. We can divide general-purpose parallel/distributed computer systems into two categories: multiprocessors and multicomputers. The main difference between them lies in the level at which interactions between the processors occur.

A multiprocessor must permit all processors to directly share the main memory. All the processors address a common main memory space. In a multicomputer, however, each processor has its own memory space, and sharing between the processors occurs at a higher level as with a complete file or data set. A processor cannot directly access another processor's local memory.

Multiprocessors can be further divided as tightly coupled and loosely coupled. In a tightly coupled system, the main memory is situated at a central location so that the access time from any processor to the

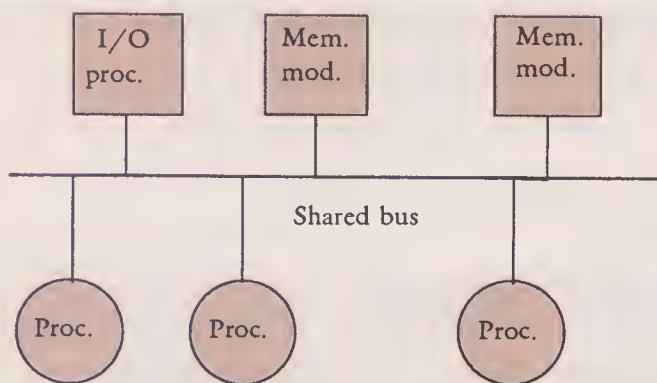
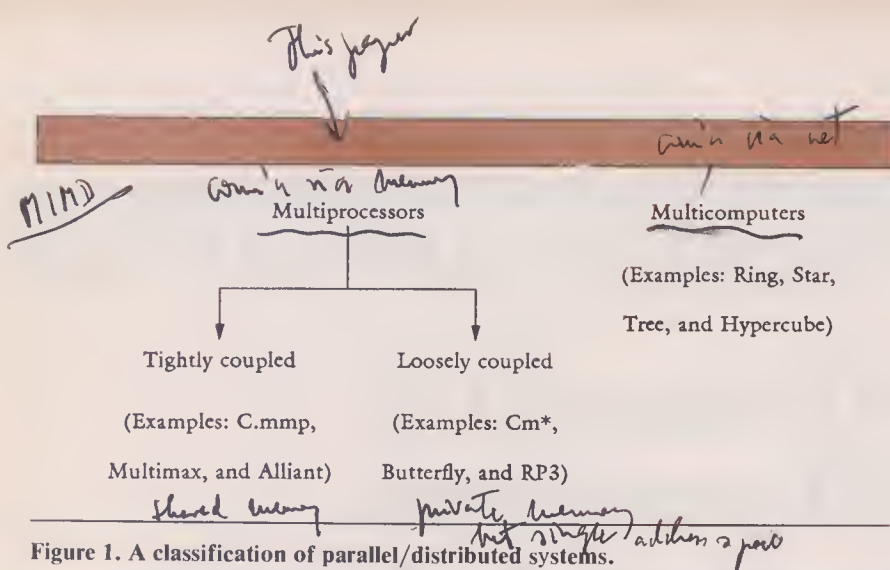


Figure 2. A single shared bus structure.

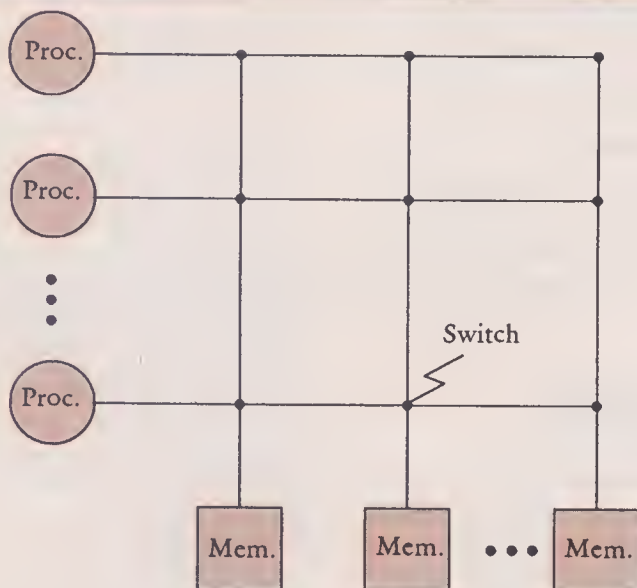


Figure 3. A crossbar interconnection network.

memory is the same. In addition to this central memory (also called main memory, shared memory, global memory, etc.), each processor might consist of some local memory or cache. The C.mmp of Carnegie Mellon University, the Multimax of Encore Computer, the FX of Alliant, and the Balance series of Sequent Corp. are examples of such tightly coupled multiprocessors.

In a loosely coupled system, the main memory is partitioned and attached to the processors, although the processors share the same memory address space. A processor can directly address a remote memory, but the access time is much higher compared to a local memory access. As a result, partitioning and allocation of program segments and data play a crucial role in the overall performance of an application program. The Cm* of CMU, the Butterfly machine of BBN Laboratories, and the RP3 of IBM are examples of such architectures.

As mentioned previously, the memory in a multicomputer is not shared. The interaction between the processors relies on message passing between the source and destination processors (nodes). The message passes over a link that directly connects two nodes and might have to pass through several such nodes in a store-and-forward manner before it reaches its destination. Therefore, each interaction involves a lot of communication overhead, and only those applications that need less interprocessor communication are well suited to multicomputers.

The multicomputers are usually based on topologies such as ring, tree, star, hypercube, etc. Hypercube machines such as Intel's iPSC are commercially available. Based on the description above, a classification of parallel/distributed computers appears in Figure 1. The classification does not include array and pipelined computers and local area networks. This is because array or pipelined computers are part of parallel processing but not distributed processing and, similarly, LANs are part of distributed processing but not parallel processing.

Essentially, our classification is valid for multiple instruction stream, multiple data stream computers. This article will concentrate solely on multiprocessor INs. A discussion of the performance of multicomputer interconnection networks can be found in Reed and Grunwald.¹

Multiprocessor IN topologies. A multiprocessor organization is defined in terms

of the IN used. The performance of a multiprocessor rests primarily on the design of its IN. A shared-bus interconnection, shown in Figure 2, is the least complex and most popular among manufacturers. The Multimax and Alliant are examples of such multiprocessors. The shared bus does not allow more than one transfer between the processors and memories at a time. A large number of processors means a long wait for the bus.

On the other hand, a crossbar, as used in C.mmp and depicted in Figure 3, supports all possible distinct connections between the processors and memories simultaneously. Unfortunately, the cost of such a network is $O(NM)$ for connecting N inputs and M outputs. For a system with hundreds of processors and memories, the cost of such an IN is prohibitively high.

In terms of cost and performance, multistage interconnection networks (MINs) and multiple-bus networks achieve a reasonable balance between those of a shared bus and crossbar. MINs and multiple-bus networks are depicted in Figures 4 and 5, respectively, and will be described in later sections. Then, we will investigate the performance of these networks. A shared bus is essentially a special type of multiple-bus IN with the number of buses equal to one.

Classification of INs

An IN is a complex connection of switches and links that permits data communication between the processors and memories. Depending on the timing philosophy, switching mode, and control strategy, each set of topologically equivalent networks can have different operational characteristics giving rise to different system behaviors. These operational characteristics also necessitate different methodologies to be used in IN performance evaluation.

Timing philosophy. Timing philosophy is one of the most important attributes characterizing a communication system. Basically, there are two types of possible timing schemes in a system: synchronous and asynchronous.

Synchronous control techniques are well understood and widely used in computer system designs. They are characterized by the existence of a central, global clock that broadcasts clock signals to all devices in a system so that the entire system operates in a lock-step fashion.

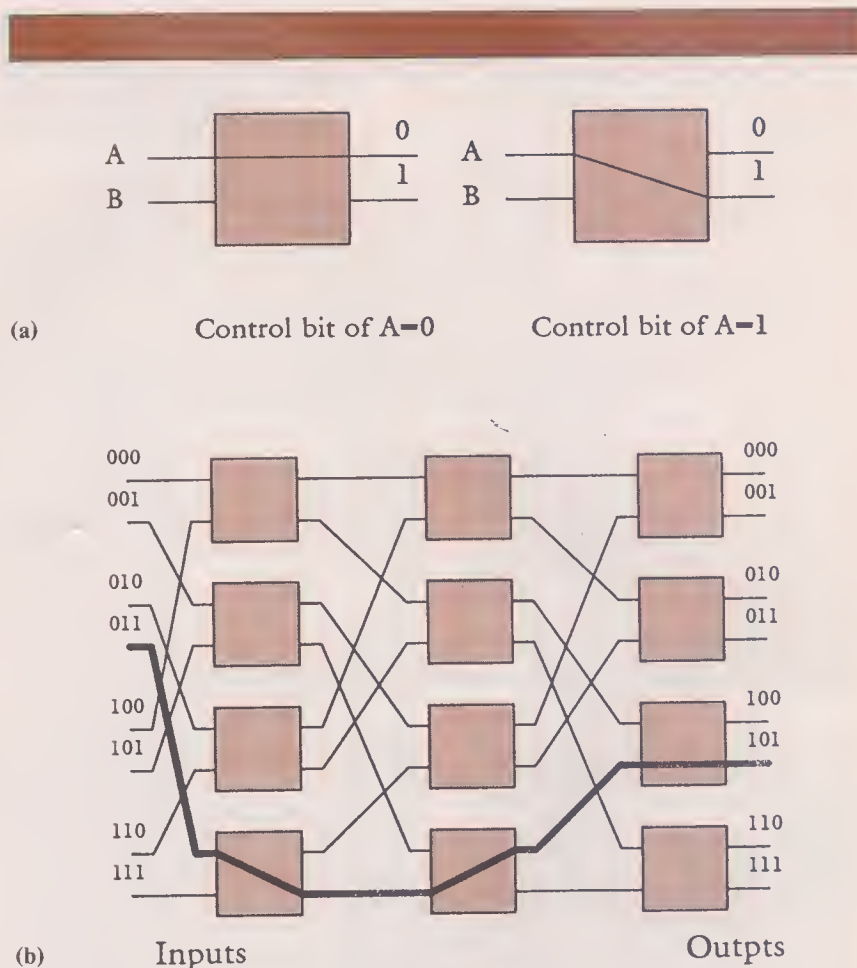


Figure 4. Operation of a 2×2 switch in 4a, and an 8×8 omega network in 4b.

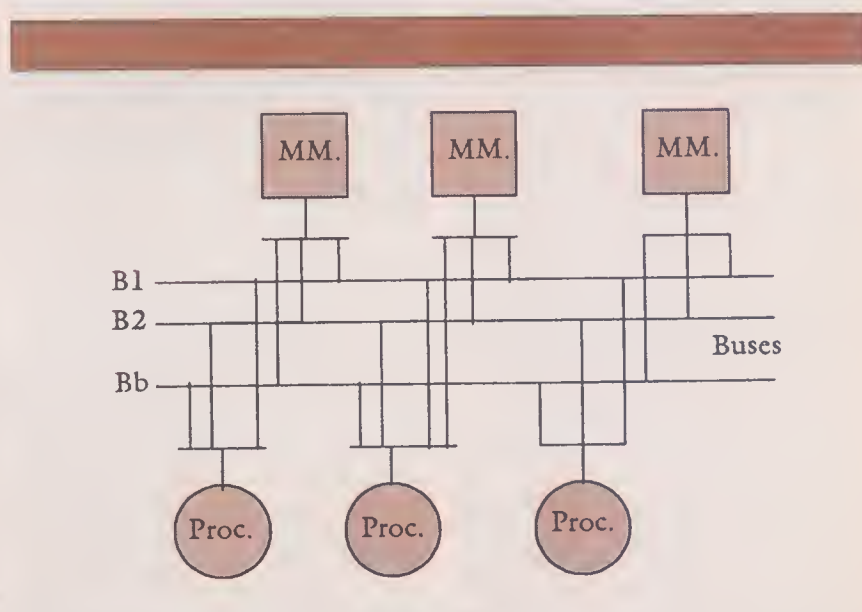


Figure 5. A multiple-bus multiprocessor system.

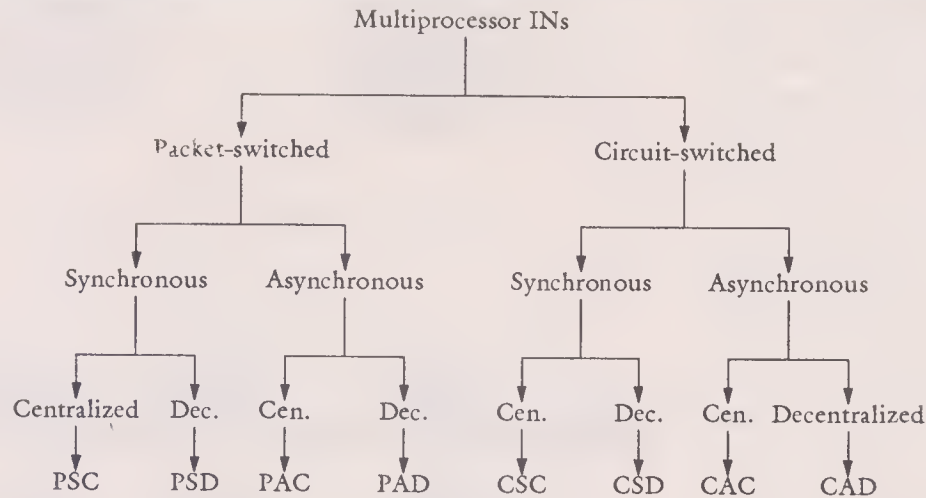


Figure 6. A classification of multiprocessor interconnection networks.

Asynchronous techniques, on the other hand, operate without a global clock. The communications among operational units in the system are performed by means of interlock hand shaking. As a result, they have good expandability and modularity, but are difficult to design.

Switching methodology. There are basically two major switching methodologies: packet switching and circuit switching. In packet switching, a message is broken into small packets transmitted through the network in a "store-and-forward" mode. Thus, a packet experiences a random delay at each switching point, depending on the traffic in the network along its path to the destination.

Conversely, circuit switching actually establishes a physical path between a source and a destination. A time delay is needed when the path is being established. Once the path is completed, it is held for the entire data transmission. In general, circuit switching is much more suitable for long messages, and packet switching is more efficient for short messages.

Control strategy. Control strategy mainly concerns the way control signals direct the dataflow generated in a network. In a centralized control scheme, all the control signals come from a single source. Obviously, the central controller creates a system bottleneck and directly affects the performance and reliability of the entire system. The design of this central con-

troller must be very complex to retain good system performance. These drawbacks can be avoided through the use of distributed control strategies in which a small controller is associated with each component of the system. In multiprocessor applications, control of crossbar networks is usually centralized and control of MINs is usually decentralized. Multiple-bus IN control can be either centralized or decentralized.

Based on the operational characteristics above, INs can be classified into eight different categories for a given topology. The detailed classification scheme is shown in Figure 6. For example, PSC means a packet-switched, synchronous, centrally controlled IN. Together with the topology, these three operational characteristics define an IN. We will examine the performance models of the INs based on this classification scheme.

Basic terminologies for performance evaluation

Before we describe performance analyses of different INs, we need to define several terms. Many performance parameters are applicable for INs. Memory *bandwidth* (*BW*) is the most common performance parameter used in analyzing a synchronous IN in a multiprocessor. It is defined as the mean number of active memory modules in a transfer cycle of the IN. In this case, the term "active" means a processor is successfully performing

memory operation (either read or write) in that memory module. *BW* also takes into account the memory access conflicts caused by the random nature of the processors' requests.

Another parameter often used in synchronous analysis, *probability of acceptance* (P_A), is defined as the ratio of expected bandwidth to the expected number of requests generated per cycle.

In asynchronous operation, the *throughput* (*Thr*) of a network is defined as the average number of packets delivered by the network in unit time. In a multiprocessor IN, throughput is the mean number of memory access completions per unit time.

Processor utilization (P_u) is also used as a performance measure and is defined as the expected value of the percentage of time a processor is active. A processor is said to be active when it is doing internal computation without accessing the global memory. Processing power is a simple extension of P_u , which is the sum of processor utilizations over the number of processors.

Other performance parameters can be easily related to the parameters above by applying Little's Law.² Moreover, P_u , *BW*, and *Thr* can also be related as

$$P_u = \frac{BW}{N\lambda T}$$

$$P_u = \frac{Thr}{\lambda}$$

where N is the number of processors, T is the time taken for a memory read or write operation, and λ is the memory request rate.

Analytical modeling is a cost effective technique used to study the performance of a computer system. However, any real system is too complex to be modeled exactly.

To make an analytical model tractable, certain approximation assumptions are necessary. Most of the IN analyses assume identical processors and a *uniform reference model*. The URM implies that, when a processor makes a memory request to the global memory, the request will be directed to any one of M memory modules with the same probability $1/M$. That is, the destination address of a memory request is uniformly distributed among M memory modules. This assumption provides us with the symmetric property, significantly simplifying the modeling.

If the memory system is M -way interleaved, this assumption also represents the program behavior reasonably accurately. When the main memory is not interleaved, there is a locality of reference and a favorite memory assumption³ is more accurate.

The *request rate* of a processor identifies how often a processor accesses global memory. This indirectly reflects the average execution time of an instruction.

In synchronous systems, the request rate can be specified by a probability that a processor generates a memory request at the beginning of a cycle. In asynchronous systems, on the other hand, a memory request could be generated at any instant in time since there is no global clock. However, an exponential thinking time for a processor is commonly assumed, which means that the duration between the completion of a request and generation of the next request to the global memory is an exponentially distributed random variable.

The *request independence assumption* (also called Strecker's approximation⁴) in a synchronous system analysis states that a memory request generated in a cycle is independent of the requests of the previous cycles. In reality, this is not true because a request that was rejected in the previous cycle will be resubmitted in the current cycle. However, as we shall see, this assumption simplifies the analysis to a great extent while keeping the results reasonably accurate.

Performance of crossbar interconnection networks

A crossbar interconnection network is an array of individually operated contact pairs in which there is one pair for each input-output combination, as shown in Figure 3. A crossbar network with N inputs and M outputs is referred to as an $N \times M$ crossbar. As long as there is no memory interference among a set of memory requests generated by the processors (that is, no two or more processors request the same memory module), all connections can be established at the same time. Thus, all memory accesses can proceed simultaneously.

But this capability comes at a high switching cost, which is $O(NM)$. Although the crossbar network can provide all simultaneous connections, memory bandwidth is much less than its actual capacity. This reduction is due to the memory interference caused by the random nature of the memory requests in a multiprocessor environment. Therefore, the performance analysis of a crossbar network becomes the analysis of memory interference.

The literature⁵ contains a number of memory interference models for centralized, synchronous, and circuit-switched crossbar systems. In most of these models, system operations are approximated by stochastic processes as follows: At the beginning of the system cycle, a processor selects a memory module at random and makes a request to access that module with some probability p . If more than one request is made to the same memory module, the memory controller will choose one at random, and the rejected processors will retry in the next cycle. The behavior of the processors is considered independent and statistically identical, as is the behavior of the memory modules.

Bhandarkar⁴ studied the memory interference problem in detail in which several discrete Markov chain models were developed. In these models, a memory module is characterized by its cycle time t_c , which consists of an access time t_a , followed by a rewrite time t_w . Processor behavior is modeled as an ordered sequence, consisting of a memory request followed by a certain amount of execution time t_p . The processing time t_p is measured from the time data were obtained from the previous

request to the time the next request is issued to the memory.

In real systems, the processor can start execution when the memory is in its rewrite cycle. So, when $t_w = t_p$, the situation would be equivalent to the case where the processor generates a memory request at the beginning of each memory cycle. In this study, an exact model for the case $t_p = t_w$ and with URM was presented. However, the model becomes very unwieldy for a large number of processors and memory modules.

The complexity of the memory interference model is simplified if one assumes that a blocked processor discards the request and generates a new independent request at the start of the next cycle (request independence assumption). For a system with N processors and M memory modules, if a processor generates a request with probability p in a cycle directed to each memory with equal probability (URM), then the memory bandwidth is given by Strecker⁴ as

$$BW = M(1 - (1 - \frac{p}{M})^N) \quad (1)$$

A simple explanation of this formula is as follows: Since p/M is the probability that a processor requests a particular memory module, $[1 - (p/M)]^N$ is the probability that none of the N processors requests the memory module in a particular cycle. Subtracting this term from 1 gives the probability that at least one request to this memory is issued. Multiplying by M yields the expected number of distinct memory modules being requested in a cycle and hence the bandwidth. The maximum percentage of error with this approximation is limited to 8 percent for $M/N > 0.75$.⁴ As a result, this simple formula, Equation 1, is widely used for predicting the performance of crossbar networks. The accuracy can be further increased by a "rate adjustment" technique⁶ where the input request rate is adjusted upward to take into account the resubmission of rejected requests. Yen⁶ provides a comparison of various memory interference models for synchronous crossbars.

The model described above assumes a URM. However, as mentioned previously, the distribution of memory requests in real systems depends on program behavior, and such distributions are not necessarily uniform. Bhuyan³ has examined this nonuniform reference problem by introducing the concept of favorite memory of a processor. The memory module requested most often by a processor is

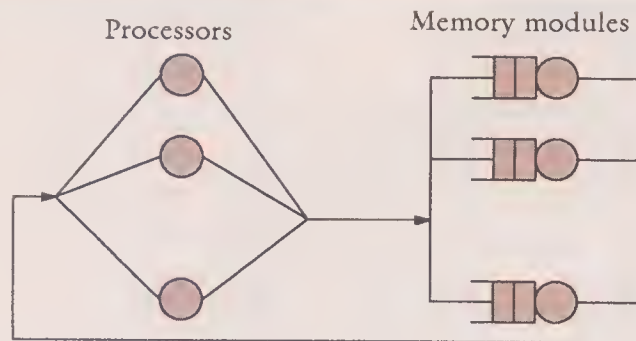


Figure 7. A queuing model for asynchronous crossbar multiprocessors.

called the favorite memory of the processor. Let m represent the probability with which a processor addresses its favorite memory given that the processor generates a request in a cycle. Then, the memory bandwidth for an $N \times N$ crossbar-based multiprocessor is given by

$$BW = \frac{N}{N-1} [1 - (1-pm)(1-p)^{N-1}] \quad (2)$$

Solutions for favorite memory cases are also provided for $M \leq N$ and $M \geq N$.⁷ By substituting $m = 1/M$, the analysis reduces to that of URM (Equation 1).

In the descriptions above, we have considered only circuit-switched and synchronous systems. The analysis of asynchronous circuit-switched systems can be done by assuming a random period of processor thinking time and memory access time. The processors are then modeled by a set of delay servers and memory modules by a set of first-come, first-serve (FCFS) queues, as shown in Figure 7. This figure depicts a well-known closed queueing network in performance evalu-

ation, and efficient algorithms to solve this network exist.²

Because the crossbar network is a single-staged network (that is, every input and output is connected by a single switching element), packet switching makes no difference from circuit switching from a performance point of view. Similarly, two control strategies result in the same system behavior. Thus, we need not consider them separately. Table 1 summarizes the different analytical techniques of the crossbar system and their accuracy, workload representations, performance metrics, and computational costs.

Analyses of multistage interconnection networks

As stated previously, the cost of a crossbar network is too high to be practical for building large multiprocessor systems. As an alternative to the crossbar network, multistage interconnection networks

(MINs) have assumed importance in recent times. The main advantage of these networks is their cost-effectiveness. They allow a rich subset of one to one and simultaneous mappings of processors to memory modules, while reducing the hardware cost to $O(N \log N)$ in contrast to $O(N^2)$ for crossbar networks.

An $N \times N$ MIN connects N processors to N memories. For N a power of two, it employs $\log_2 N$ stages of 2×2 switches with $N/2$ switches per stage. Each switch has two inputs and two outputs. The connection between an input and an output is established depending on a control bit c provided by the input. When $c = 0$, the input is connected to the upper output, and when $c = 1$, it is connected to the lower output, as shown in Figure 4a.

An omega network^{5,8}, shown in Figure 4b, is characterized by a perfect shuffle interconnection preceding every stage of switches. The requesting processor generates a tag that is the binary representation of the destination. The connection of a switch at the i th stage is then accomplished by the i th bit of this binary tag counted from the most significant bit.

The connection between input 3 and output 5 (101_2) is shown by a bold line in Figure 4b. This self-routing property of a MIN avoids the need for a central controller, making it very suitable for multiprocessors. Thus, the performance discussions presented in this section will concentrate solely on the decentralized control scheme.

Many significant MINs, such as Banyan, generalized cube, base line, etc.,⁸ have been proposed. However, most of these networks are similar except for the interconnection between the adjacent stages.

The switch size in an MIN need not be restricted to 2×2 . In fact, the Butterfly parallel processor connects N inputs to N

Table 1. Summary of crossbar analyses.

	Synchronous crossbar		Asynchronous crossbar
	Discrete Markov chain ⁴	Probabilistic with independence assumption ^{3,4}	Queueing network ²
Analysis technique	Discrete Markov chain ⁴	Probabilistic with independence assumption ^{3,4}	Queueing network ²
Workload representation	Request rate	Probability of request	Think time
Performance parameters	BW or P_A	BW or P_A	P_u or P_w
Accuracy	Exact	Good	Exact
Computational cost	Very high	Low (closed form formula)	Moderate

outputs using 4×4 crossbar switches and $\log_4 N$ stages with $N/4$ switches per stage. A delta network can connect $M = a^n$ inputs to $N = b^n$ outputs through n stages of $a \times b$ crossbar switches.⁹ The generalized shuffle network (GSN) is capable of connecting any $M = m_1 * m_2 * \dots * m_r$ inputs to $N = n_1 * n_2 * \dots * n_r$ outputs through r stages of switches.⁷ The i th stage employs $m_i \times n_i$ crossbar switches and is preceded by a generalized shuffle interconnection that is essentially a superset of the omega and delta interconnections. This is the most generalized version of an MIN that allows different input and output sizes, and all the other networks can be obtained by choosing the m_i s and n_i s, appropriately. For example, when $m_i = a$, $n_i = b$ for all i s, it is a delta network; $m_i = n_i = 2$ for all i s gives an omega network; $r = 1$ gives a crossbar; and $M = M * 1$ and $N = 1 * N$ provides a shared-bus connection.

The advantages of MINs were widely recognized by researchers, and a lot of research projects started at universities and industries. Examples of university projects include TRAC (the Texas Reconfigurable Array Computer) at the University of Texas at Austin, Pasm (partitionable single instruction, multiple data [SIMD]), multiple instruction, multiple data [MIMD]) at Purdue University, Ultra-Computer at New York University, and Cedar at the University of Illinois at Urbana-Champaign. RP3 is a notable industry project at IBM, and Butterfly is a successfully marketed product by BBN Laboratories.

As these projects were starting, a serious drawback of the MINs surfaced. There is only one path from an input to an output. It was necessary to incorporate some fault-tolerance into these networks so that at least a single fault in a switch or a link could be tolerated. This has given rise to an abundance of research during the past few years devoted to the design and evaluation of fault-tolerant MINs. Adams¹⁰ contains a survey and comparison of such fault-tolerant networks. The evaluation techniques for basic MINs are explained below, but can be extended to fault-tolerant MINs.

Patel⁹ suggested a probabilistic approach to analyze the delta network based on URM and the request independence assumption. Assume a delta network of size $a^n \times b^n$ constructed from $a \times b$ crossbar modules. Each stage of the delta network is controlled by a distinct destination digit (in base b) for setting of

individual $a \times b$ switches. Since the destinations are independent and uniformly distributed, the requests at any $a \times b$ modules are independent and uniformly distributed over b different destinations. In addition, the switches at a particular stage behave similarly. Therefore, Equation 1 can be applied to any switching element in the delta network.

The expected number of requests that pass to the b outputs is obtained by setting $N = a$ and $M = b$ in Equation 1. Dividing this number by b gives us the probability of request on any of the b output lines of an $a \times b$ switch as a function of its input probability. Since the output of a stage is the input of the next stage, one can recursively evaluate the output probability of any stage starting at stage 1. If p_i is the probability that there is a request at the output of a switch at stage i , then

$$p_i = 1 - (1 - \frac{p_{i-1}}{b})^a \quad (3)$$

for $1 \leq i \leq n$. In particular, the output probability of the final stage determines the bandwidth of a delta network, that is, $BW = p_n b^n$. This analytical technique has been widely used to evaluate various MINs.

Bhuyan³ extended the analysis to favorite-memory cases. For $N \times N$ networks, with $N = a^n$, the processors are defined to be connected to their favorite memories when all the switches are straight connected, that is, input i of a switch is connected to the output i of the switch. In an omega network memory, MM_j becomes the favorite memory of processor P_j . Let q_{i-1} be the probability that there is a favorite request to the input at stage i . In an $N \times N$ ($N = a^n$) delta network,

$$p_i = \frac{1 - (1 - p_{i-1} q_{i-1}) (1 - p_{i-1} \frac{1 - q_{i-1}}{a - 1})}{1 - p_{i-1} q_{i-1}} \quad (4)$$

About six other equations are needed to determine q_{i-1} at stage i^3 and, finally, $BW = N p_n$. The analyses above^{3,9} are valid for synchronous packet-switched MINs provided that (a) packets are generated only at the beginning of the network cycle and (b) switches do not have buffers (are unbuffered) so that packets are randomly chosen in case of conflicts and unsuccessful packets are lost.

The above analyses presented a recurrence relation for the performance of unbuffered networks, but not a closed-form solution. Kruskal and Snir¹¹ obtained an asymptotic expression for the output request probability of a stage for an

unbuffered delta network. Let p_m denote the probability that there is a packet on any particular input at the m th stage of a square MIN composed of $k \times k$ switches. Through some algebraic manipulations, Kruskal and Snir¹¹ approximated the asymptotic formula for p_m as

$$p_m = \frac{2k}{(k-1)m + \frac{2k}{p}} \quad (5)$$

where p is the probability of request generation by a processor. From this expression, one can see that the probability that a message is not deleted is inversely proportional to the number of stages in the network. The solution for an unbuffered network by Kruskal and Snir has been shown to be a strict upper bound in the throughput of a delta network. For buffered networks, Kruskal and Snir assume an infinite buffer associated with each output of a switching element. In each cycle, a random number of packets join an output queue without any packet being lost. This random number has a Bernoulli distribution, since all incoming packets from the inputs of the switch have an equal probability (for URM) of going to that output. The average transit time through the network can be derived by means of the $M/G/1$ queueing formula.¹¹

Dias and Jump¹² have studied buffered delta networks by means of petri nets, which were introduced first as a useful graphical tool for the precise description of the system operations and as a modeling technique that permits easy evaluation of the performance of small systems.

These graph models have recently become very popular for the representation of distributed computing systems because of their ability to clearly describe concurrency, conflicts, and synchronization of tasks. However, the complexity of petri nets increases exponentially with the increase in system size. With this modeling technique, 14 distinguishable states of a (2×2) switch exist with a single buffer between stages.¹² The state transition tables and the probabilities in each state are derived. The steady state throughput and turnaround time (network delay) are obtained by iterating through use of the transition tables and probability equations. The results of analysis and simulation indicate that buffering produces a considerable improvement in the performance of these networks.

All the analyses above pertain to synchronous circuit-switched or packet-switched environments. For large MINs,

Table 2. Summary of MINs analyses.

	Synchronous MIN without buffer	Synchronous MIN with infinite buffers	Synchronous MIN with finite buffers	Asynchronous MIN with finite buffers
Analysis technique	Probabilistic with independence assumption ^{3,9}	$M/G/1$ queue with infinite buffers ¹¹	Petri net ¹²	Multiple chain MVA ⁵
Workload representation	Request rate	Request rate	Processor think time	Processor think time
Performance parameters	BW or P_A	Queueing delay or transit time	Throughput or turn-around time	Throughput or response time
Accuracy	Good	Fair	Good	Good
Computation cost	Low for recurrence solutions	Closed form formula	High	Moderate

controlling the network operation from a central global clock is difficult. Hence, asynchronous designs should be considered for large multiprocessor systems.

The second disadvantage with the above analyses is that they do not incorporate the waiting time of a processor for completion of the memory access, but assume continuous Poisson arrival at the input side.

The third disadvantage is the assumption of uniform or favorite memory access. Memory reference patterns are highly program dependent and could be arbitrary.

To overcome these drawbacks, we have recently developed⁵ a closed queueing network model and mean value analysis (MVA)² for the MINs under asynchronous packet-switched operation. Modeling asynchronous circuit-switched MINs seems very difficult because of the simultaneous possession of switches and links by unsuccessful requests. Table 2 lists the different analyses of MINs described in this section.

Performance analyses of multiple-bus systems

Most commercial systems containing more than one processor employ a single shared bus as shown in Figure 2. This interconnection scheme is well known as being inexpensive and easy to implement. But when the system size is large, a single bus becomes a severe system bottleneck.

A natural extension is to employ several

buses instead of a single one to increase the bandwidth and fault tolerance at moderate cost. Recently, the multiple-bus interconnection scheme has drawn considerable attention from many computer scientists and engineers. In an $N \times M \times B$ multiple-bus multiprocessor system, all the N processors and M memory modules are connected to all the B buses.

Unlike a crossbar or multistage network, the multiple-bus configuration offers high reliability, availability, and easy incremental system growth. Higher reliability is obvious because, in case of a bus failure, $(B - 1)$ distinct paths still exist between a processor and a memory. However, when the number of buses is less than the number of memory modules or the number of processors, bus contention can arise. As a result, the performance analysis of a multiple-bus system involves modeling the effects of bus conflicts and memory interference.

Many researchers have studied the performance of synchronous, circuit-switched, and centrally controlled multiple-bus multiprocessor systems through analysis and simulation.^{5,13} The memory bandwidth of the system increases with an increase in the number of buses. But, for all practical purposes, a few buses might be sufficient.

In a synchronous system, all the events occur at the beginning of a system cycle. Therefore, the system can be modeled by means of a discrete Markov process. Bhuyan has developed a combinatorial and probabilistic approach to derive an

equation for the BW of such multiple-bus multiprocessor systems.^{5,13} His analysis is based on the URM and request independence assumptions.

With these assumptions in mind for a given set of memory requests, knowing the number of ways in which the requests are distributed among the memory modules is easy. In addition, one can determine the number of ways by which the given requests are addressed to M memory modules such that x memory modules are requested and $M - x$ of them are idle.

For each value of x , Bhuyan defines a state. The BW can be computed by multiplying the probability of being in a state with the number of busy memory modules in that state. If the number of buses in a system is less than the number of memory modules, the number of busy memory modules in a cycle would be upper bounded by the number of buses. The bandwidth for an $N \times M \times B$ system is given by

$$\begin{aligned}
 BW = M & \left\{ 1 - \left(1 - \frac{p}{M} \right)^N \right\} \\
 & - \sum_{y=B+1}^N \binom{N}{y} p^y (1-p)^{N-y} \\
 & \times \sum_{x=B+1}^{t_y} \frac{(x-B)x! \binom{M}{x} S(y,x)}{M^y}
 \end{aligned} \quad (6)$$

where, $x!$ is the factorial x , $t_y = \min(y, M)$, p is the probability of request of a processor, $\binom{N}{y}$ is the binomial coefficient, and $S(y, x)$ is the Stirling number of the second kind defined as

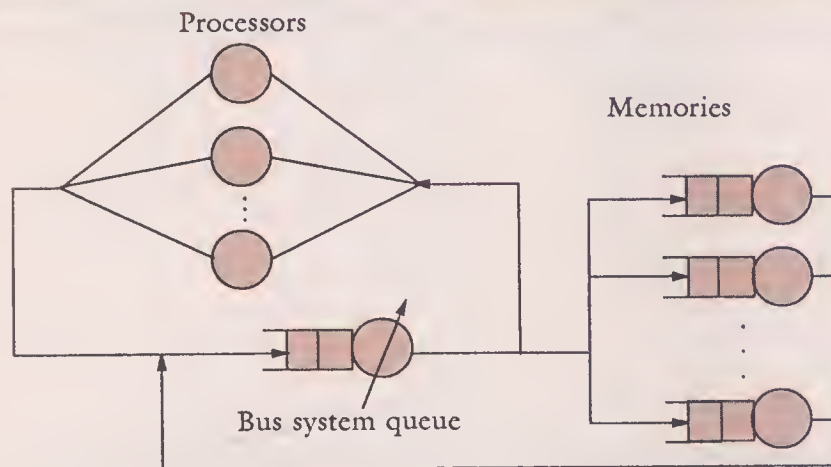


Figure 8. A queuing network model for asynchronous multiple-bus systems.

$$x! S(y, x) = \sum_{i=0}^x (-1)^i \binom{x}{i} (x-i)^y$$

The numerical results obtained from this equation are quite close to simulation results. Mudge¹³ contains a detailed review and comparison of various analyses on synchronous circuit-switched multiple-bus systems.

In asynchronous systems, memory request generation and memory access completion can occur at any point in time since there is no synchronization by a clock. Therefore, the system can be modeled with a continuous stochastic process. Each shared resource, such as a bus or a memory module, is considered a queuing service center.

Because of circuit switching, the processor that issues a memory request holds a system bus while accessing the main memory. Thus, two system resources, bus and memory, are simultaneously held by a memory operation. This simultaneous resource possession phenomenon makes the analysis nontrivial.

One study¹⁴ uses the *flow equivalence* technique to approximately solve the queuing network. The bus and memory subsystem, called aggregate, is replaced by a single flow equivalent service center (FESC).² The model has been compared with simulation results, and the agreement is quite good over a wide range of bus load.¹⁴

All the studies above consider only centrally controlled, circuit-switched

multiple-bus systems. In circuit switching, a device will occupy the bus for the entire duration of data communication once the device is granted use of a bus.

For instance, a processor in a read operation will occupy the bus during the time it is sending a request, performing a memory operation, and receiving the requested data. The result will be the waste of a significant fraction of bus bandwidth because of a mismatch between the speeds of the processing unit, the bus, and the memory unit.

All these facts serve to demonstrate the attractiveness of the packet-switching approach. Encore's Multimax is an example of a multiprocessor employing a packet-switched shared bus between processors and memories.

Yang studied¹⁵ packet-switched multiple-bus systems where analytical models were developed for both synchronous and asynchronous timings. In the synchronous case, both centralized and decentralized controlled schemes are treated equally, since all the events in the system occur at the beginning of a cycle regardless of the control strategies used. A discrete probabilistic approach is applied to analyze such systems.

The model has been based on a decomposition technique that considers simple analysis of a set of single-server queues. The consequence of the analysis is an equation consisting of one unknown variable, P_u , processor utilization; it can be solved by using a standard numerical method.

For an asynchronous case, the queuing

network is shown in Figure 8. Processors in the system are modeled as delay servers, and memory modules are modeled as FCFS servers. The bus system is modeled as an FESC² representing B buses with a single (centralized) queue.

The routing of a packet in the network can be described as follows: A request packet generated by a processor is first put in the central server queue, waiting for an available bus. After it gains access to a bus, the packet joins one of the M memory queues. The memory module that finishes the service of a request again puts the response packet in the central server queue. From there, the response packet gets back to the requesting processor through a bus. To this point, the packet finishes one rotation through the network, and the processor resumes its background activity. The model can be solved using any standard product-form algorithm, considering the bus system queue as a load-dependent server.²

In the case of a decentralized control, the actual implementation can be either token bus or daisy chain bus. Due to the lack of central controller, the analysis of decentralized packet-switching multiple-bus systems is very complicated. The FCFS service discipline is not valid for bus system queue in this case. The service discipline depends on the position of tokens with respect to the positions of requesting devices.

The exact solution for such a system seems infeasible, but one method for approximating the behavior of the system is to use hierarchical modeling tech-

Table 3. Summary of multiple-bus analyses.

	Circuit-switched synchronous	Circuit-switched asynchronous	Packet-switched synchronous	Packet-switched asynchronous
Analysis technique	Probabilistic with independence assumption ^{5,13}	Queue network with infinite buffers ¹⁴	Probabilistic queueing analysis ¹⁵	Queueing network with FESC ¹⁵
Workload representation	Request rate	Processor think time	Request rate	Processor think time
Performance parameters	BW or P_A	Throughput or P_u	P_u	Throughput or P_u
Accuracy	Fair	Good	Good	Fair
Computation cost	Low	Moderate	Low, but iteration	Moderate

Table 4. Summary of hardware features of the three INs.

	Crossbar	MINs	Multiple-bus
No. switches or connections	$N \times M$	$M \log N$	$B \times (N + M)$
Load of buses	N	1	B
No. of wires	M	N	B
Arbiter	M 1-of- N arbiters	$M \log N$ 1-of-2 arbiters	1 B -of- M and M 1-of- N arbiters
Fault-tolerant and expansion	Fair	Poor, but fair with additional hardware	Good

niques.¹⁵ As in the previous case, the bus system is represented by FESC (Figure 8) with the service rate obtained by shorting out the processors and the memory modules. The model is then solved by using the mean value analysis algorithm.²

Numerical results obtained from the models have shown that packet switching reduces the communication bottleneck of shared bus systems.¹⁵ Table 3 summarizes analyses of different categories of multiple-bus systems.

Comparison and discussions

As indicated in the previous sections, the three types of interconnection networks possess different hardware features and different system performances. In this section, we will look into these differences

quantitatively. In particular, we will compare the hardware cost and system performance of the three interconnection networks.

Table 4 lists selective hardware features of the three networks. As we already know, the number of switching elements used in an $N \times M$ crossbar is $N \times M$ in contrast to $N \log N$ of MINs. The number of connections necessary in an $N \times M \times B$ multiple-bus system is proportional to $B(N + M)$. Since each bus in the multiple-bus system needs to drive $N + M$ modules, the bus load is proportional to $N + M$, while the bus load of an MIN is one due to the one-to-one connection.

All three of these networks require certain types of arbiters to resolve the request conflicts. In a crossbar network, M N -users 1-server arbiters are necessary, each of which selects one of up to N outstanding requests for a memory during a mem-

ory cycle. The MINs, on the other hand, require two 2-user 1-server arbiters for each switching element for $(N/2) \log_2 N$ switches.

In a multiple-bus system, an M -users B -servers arbiter is needed to assign the B buses to the outstanding requests. Once a bus is granted to a memory, only one of the processors that requests the memory can proceed while the others, if any, are delayed. This choice is implemented by an N -users 1-server arbiter. Thus, a multiple-bus system requires $M + 1$ arbiters, one arbiter of the M -users B -servers type and M arbiters of the N -users 1-server type.

Expandability and reliability are two other very important hardware features. In this context, a multiple-bus system shows its advantages over the other two because of its reconfigurability and multiple-data paths between every processor and memory. It can still operate in a

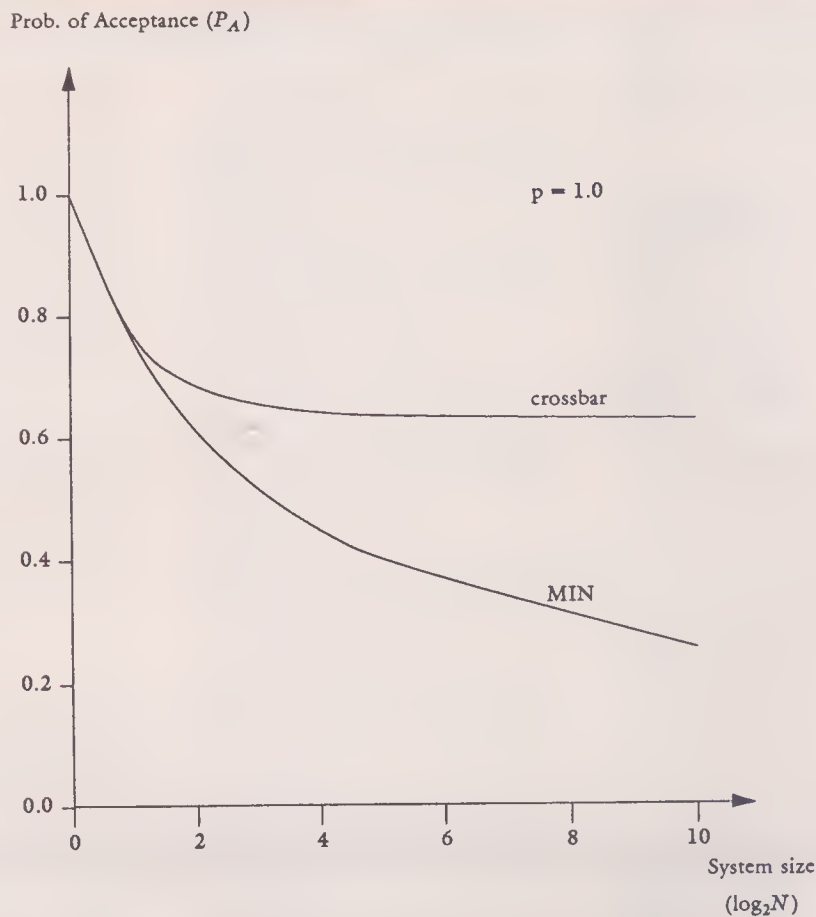


Figure 9. Probability of acceptance as a function of system size for synchronous circuit-switched systems.

degraded mode after the failure of a subset of the buses.

A lot of research has gone into the reconfigurable and fault-tolerant MINs.¹⁰ The fault-tolerance of a MIN can be realized by adding additional hardware such as extra stages or duplicated data paths. It is easier to reconfigure a crossbar than an MIN. In case of a fault, a particular row or a column in Figure 3 can be removed and the network can operate in a degraded mode.

Next, we'll consider the performance of the three interconnection networks based on the analytical models described in the previous sections. Figure 9 shows the probability of a memory request being accepted, $P_A = BW/p \cdot N$, as a function of system size for synchronous circuit-switched systems with $p = 1.0$.

Two curves, one for crossbar and one for MIN, are plotted according to Equa-

tion 1 and Equation 3, respectively. The difference between the two curves increases as the system size grows. The probability of acceptance in the crossbar system remains constant when the system size becomes very large. However, in the case of MIN, P_A keeps decreasing as the system size increases.

Figure 10 shows the memory bandwidth of 16×16 synchronous circuit-switched multiprocessor systems. The horizontal axis represents a processor's probability of request. As we can see, the single bus performs worse since it gets saturated very quickly and the BW can never exceed 1. The BW increases as the number of buses increases, as shown in the figure.

Figure 11 shows the comparison of synchronous packet-switched networks. In this figure, processor utilizations are plotted against the probability of request for a 16×16 multiprocessor. Processor cycle

times, bus transfer delay, as well as the total ideal packet transfer time between an input and an output of an MIN are assumed to be fixed at the system cycle time. The memory access time is assumed to take four system cycles (similar to Encore's Multimax) for all three systems. During a memory access, the processor that issued the memory request remains idle until the memory access is finished.

The memory request rate of a processor as seen by an interconnection network is adjusted⁶ in plotting the processor utilization in Figure 11. The performance difference between various networks is not as pronounced as in Figure 10. This is due to the packet-switched operation of the INs. Additionally, a multiple-bus-based multiprocessor system with only four buses can achieve almost the same performance as that of the crossbar system while reducing the hardware cost significantly.

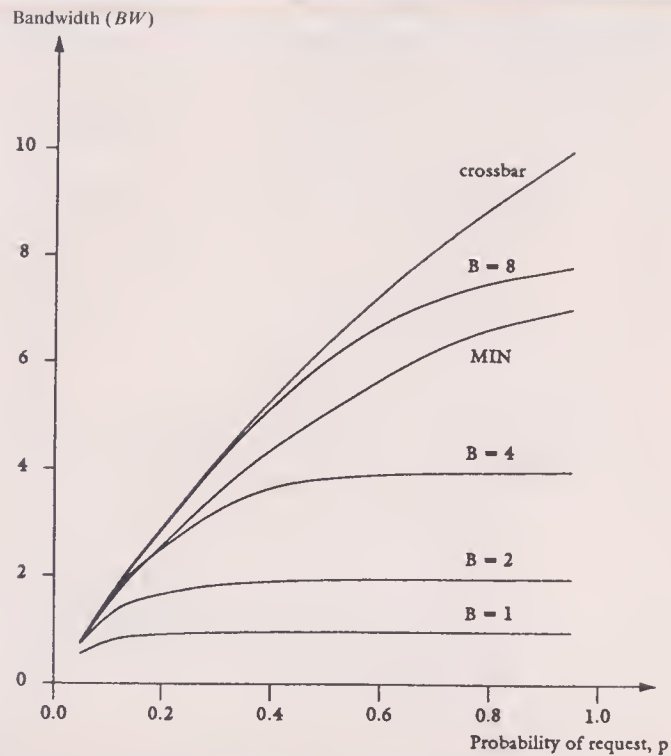


Figure 10. Memory bandwidth as a function of probability of request for 16×16 synchronous circuit-switched systems.

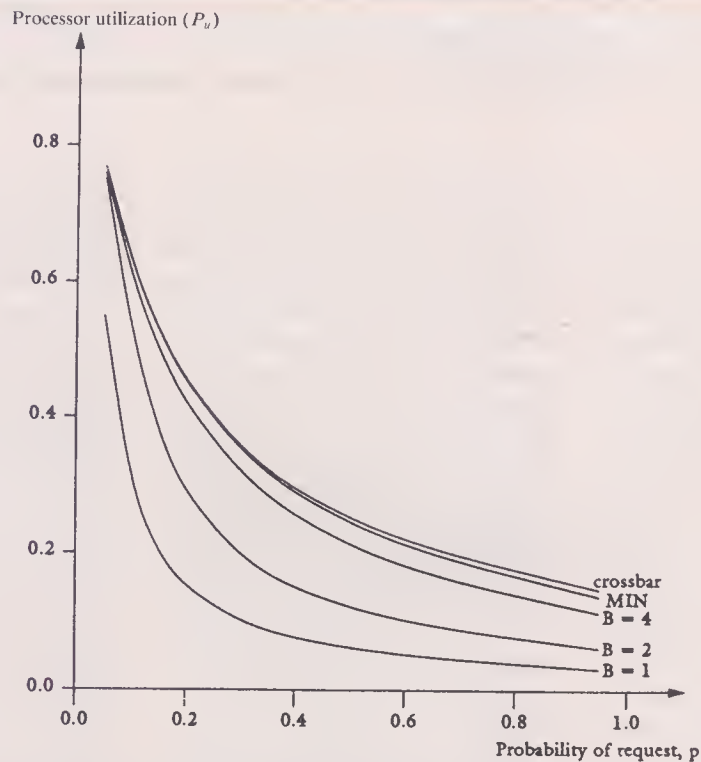


Figure 11. Processor utilization as a function of probability of request for 16×16 synchronous packet-switched systems.

All the comparisons above apply to synchronous systems based on system cycle. We have intentionally avoided comparing asynchronous systems because their performance is so dependent on the input parameters that choosing the wrong parameters might give rise to the wrong conclusions. However, the analytical techniques that can be applied to evaluate those systems are given in this article. The information provided here is useful for predicting the approximate performance of an IN structure before its design and implementation. Further references and a more detailed survey on the performance evaluation of multiprocessor INs can be found in Bhuyan.⁵

It seems that enough research has already been done in evaluating INs in isolation. We strongly feel that more work is needed at the system level that includes the IN as a major component. For example, evaluation of multiprocessor systems with prefetching, bulk data read or write, and solving cache coherence with INs shows promise for future research.

Similarly, task (application) level modeling on multiprocessor architectures might produce some good insight into the trade-offs between computation versus communication, low versus large granularity, static versus dynamic scheduling, etc.

Finally, some actual measurements (traces) should be obtained on real multiprocessors and applied to the analytical models as their input parameters. □

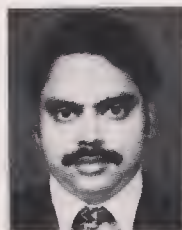
Acknowledgments

Bhuyan's research was supported by National Science Foundation grant No. MIP-8807761 and by a grant from the Louisiana Board of Regents. Agrawal's research was partly supported by US Army Research Office contract No. DAAG 29-85-K-0236.

References

1. D.A. Reed and D.C. Grunwald, "The Performance of Multicomputer Interconnection Networks," *Computer*, Vol. 20, No. 6, June 1987, pp. 63-73.
2. E.D. Lazowska et al., *Quantitative System Performance—Computer System Analysis Using Queueing Network Models*, Prentice Hall, Englewood Cliffs, N.J., 1984.
3. L.N. Bhuyan, "An Analysis of Processor-Memory Interconnection Networks," *IEEE Trans. Computers*, Vol. C-34, No. 3, Mar. 1985, pp. 279-283.

4. D.P. Bhandarkar, "Analysis of Memory Interference in Multiprocessors," *IEEE Trans. Computers*, Vol. C-24, Sept. 1975, pp. 897-908.
5. L.N. Bhuyan, "Performance Evaluation of Multiprocessor Interconnection Networks," *Tutorial Note*, ACM SIGMetrics Conf., May 1988.
6. D.W. Yen, J.H. Patel, and E.S. Davidson, "Memory Interference in Synchronous Multiprocessor Systems," *IEEE Trans. Computers*, Vol. C-31, Nov. 1982, pp. 1,116-1,121.
7. L.N. Bhuyan and D.P. Agrawal, "Design and Performance of Generalized Interconnection Networks," *IEEE Trans. Computers*, Vol. C-32, Dec. 1983, pp. 1,081-1,090.
8. Tse-Yun Feng, "A Survey of Interconnection Networks," *Computer*, Vol. 14, No. 12, Dec. 1981, pp. 12-27.
9. J.H. Patel, "Performance of Processor-Memory Interconnections for Multiprocessors," *IEEE Trans. Computers*, Vol. C-30, Oct. 1981, pp. 771-780.
10. G.B. Adams III, D.P. Agrawal, and H.J. Siegel, "A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks," *Computer*, Vol. 20, No. 6, June 1987, pp. 14-27.
11. C.P. Kruskal and M. Snir, "The Performance of Multistage Interconnection Networks for Multiprocessors," *IEEE Trans. Computers*, Vol. C-32, Dec. 1983, pp. 1,091-1,098.
12. D.M. Dias and J.R. Jump, "Analysis and Simulation of Buffered Delta Networks," *IEEE Trans. Computers*, Vol. C-30, Apr. 1981, pp. 273-282.
13. T.N. Mudge et al., "Analysis of Multiple-Bus Interconnection Networks," *J. Parallel and Distributed Computing*, Vol. 3, No. 3, Sept. 1986, pp. 328-343.
14. D. Towsley, "Approximate Models of Multiple Bus Multiprocessor Systems," *IEEE Trans. Computers*, Vol. C-35, Mar. 1986, pp. 220-228.
15. Q. Yang, L.N. Bhuyan, and R. Pavaskar, "Performance Analysis of Packet-Switched Multiple-Bus Multiprocessor Systems," *Proc. 8th Real-Time System Symp.*, Dec. 1987, CS Press, Los Alamitos, Calif., pp. 170-178.



Laxmi N. Bhuyan is an associate professor at the Center for Advanced Computer Studies at the University of Southwestern Louisiana in Lafayette. His research interests include parallel and distributed computer architecture, performance and reliability evaluation, and local area networks.

Bhuyan received BS and MS degrees in electrical engineering from the Regional Engineering College, Rourkela under Sambalpur University in India. He received a PhD in computer engineering from Wayne State University in Detroit in 1982. Bhuyan is a senior member of the IEEE, a distinguished visitor of the IEEE Computer Society, and served as guest editor of the *Computer* issue on interconnection networks in June 1987.



Qing Yang is an assistant professor in the Department of Electrical Engineering at the University of Rhode Island. His research interests include parallel and distributed computer systems, design of digital systems, performance evaluation, and local area networks.

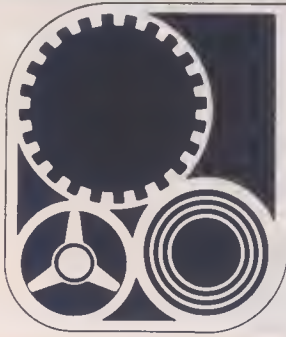
Yang received a BSc degree in computer science from Huazhong University of Science and Technology in China in 1982 and an MSc in electrical engineering from the University of Toronto in 1985. He received a PhD in computer engineering from the Center for Advanced Computer Studies, University of Southwestern Louisiana. Yang is a member of the IEEE Computer Society.



Dharma P. Agrawal is a professor of electrical and computer engineering at North Carolina State University in Raleigh. He is the group leader for the B-Hive multicomputer project at NCSU. His research interests include both software and hardware aspects of parallel and distributed processing, computer architecture, and fault tolerant computing.

Agrawal received the BE degree from Ravishankar University in India, the ME degree from the University of Roorkee in India, and the DScTech degree in 1975 from the Federal Institute of Technology in Switzerland. He is the author of the tutorial text *Advanced Computer Architecture* published by the IEEE Computer Society; a member of the *Computer* Editorial Board; an editor of the *Journal of Parallel and Distributed Computing* and the *International Journal on High-Speed Computing*; a fellow of the IEEE; a member of the IEEE Computer Society; and a recipient of the society's Certificate of Appreciation.

Readers may write to Bhuyan at the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA 70504-4330.



11th International Conference on **SOFTWARE ENGINEERING**

May 15-18, 1989 • Pittsburgh, PA, USA

Twenty Years of Software Engineering: Looking Forward, Looking Back

The term "software engineering" was coined in 1968 to denote a new discipline concerned with the technological aspects of developing and modifying computer software systems. Twenty years have elapsed since the 1968-69 NATO Workshops on Software Engineering, which are generally acknowledged to be seminal meetings. It is now appropriate to commemorate the past twenty years of software engineering and to look forward to the next twenty years.

The Conference will attempt to identify recurring themes and lasting contributions of the past twenty years and to identify the major trends in hardware, software, and society that will shape the future of software engineering.

The Conference includes a Technical Program, Tutorials, and a Tools Fair. Each track of the Technical Program includes major talks, papers, workshop reports, and panel discussions. The Tutorials precede the other activities and provide in-depth, one-day introductions to a variety of technologies and issues pertinent to software engineering. The Tools Fair is composed of an exhibition and presentation track and will provide conference attendees an opportunity to view and learn about the latest commercial and experimental software tools.

Pittsburgh ... A Well Kept Secret

Pittsburgh — one of America's most surprising cities. Long thought of as primarily an industrial center, Pittsburgh today boasts one of the nation's most striking skylines nestled between three rivers and the green, rolling hills of southwestern Pennsylvania. Since the 1950's, the city has undergone both a physical and economic rebirth. Pittsburgh is now an international corporate and high tech center that boasts Rand McNally's title of "America's Most Livable City." In Pittsburgh, you will enjoy comfortable temperatures (normal daily high in May is 70°F) while safely strolling the city's shopping and cultural districts. Riverboat cruises, hill-climbing incline cars, unique restaurants, and historical sites await your exploration.

The conference will be held in Downtown's Golden Triangle at the David L. Lawrence Convention Center and the Vista Hotel, which are linked by an enclosed walkway. This is a short walk from numerous parks, hotels, performing arts centers, major shopping facilities, and Three Rivers Stadium.

A banquet on May 17 will feature the River City Brass Band. The River City's 27 virtuoso musicians, touring since 1984, have been winning critical praise and standing ovations across the country. Often called the "Boston Pops — In Brass," the band's versatile repertory ranges from familiar folk tunes to ballads, classical transcriptions and, of course, marches. The banquet is free for delegates and \$30 for an accompanying guest.

CHAIR

Larry Druffel
Software Engineering Institute, USA

PROGRAM CO-CHAIRS

Richard Fairley
George Mason University, USA
Dines Bjorner
Dansk Datamatic Center, Denmark

TOOLS FAIR CHAIR

Grady Booch
Rational, USA

TUTORIALS CHAIR

Norman Gibbs
Software Engineering Institute, USA

LOCAL ARRANGEMENTS CHAIR

Nancy Belz
Software Engineering Institute, USA

ORGANIZED BY:

 ACM Special Interest Group
on Software Engineering

 The IEEE Computer Society
Technical Committee
on Software Engineering

 Software Engineering Institute

IN ASSOCIATION WITH:

 Association for Computing
Machinery

 Institute of Electrical and
Electronics Engineers, Inc.

 aifcet

Association Francaise pour la
Cybernetique Economique
et Technique
British Computer Society



Israel Section of the IEEE



Korea Information Science Society

 NZCS

New Zealand Computer Society, Inc.

Pittsburgh High Technology Council

 SCS

Singapore Computer Society



Software Engineers Association of
Japan

TUTORIALS • May 15, 1989

All tutorials will be held on Monday, May 15, from 9:00 a.m. to 5:00 p.m.

Tutorial 1: Formal Methods in Software Engineering

Formal methods provide a mathematical foundation for many of the intellectual activities of software development. Several different models and languages have been proposed for the specification and design of software systems. This tutorial will survey many of these models and languages, and will provide in-depth coverage of one popular method: the Vienna Development Method (VDM). The relative strengths and weaknesses of various formal specification models will be discussed using examples drawn from practical software engineering problems. A brief introduction to the specification language and principles of VDM will be given, followed by examples of its use in compiler and database areas.

Instructors: Dr. Mark A. Ardis, Software Engineering Institute and Jan Storbak Pedersen, Dansk Datamatik Center

Tutorial 2: Making Software Engineering Happen

Many of us have been frustrated by the slow rate at which software engineering practices have been adopted in industry. The problem has less to do with technology than it does with the cultural changes that result from technology transition. This tutorial presents a pragmatic, five-step "life cycle" strategy for implementing software engineering. The strategy, which has proven successful in many companies, encompasses auditing, training, selection and justification, installation, and evaluation methods.

Instructor: Dr. Roger S. Pressman, R.S. Pressman & Associates, Inc.

Tutorial 3: Software Configuration Management

Software configuration management (SCM) is the discipline of controlling the evolution of complex software systems. SCM is essential for multi-person, long-term software projects. This tutorial covers the state of the art in SCM, with special emphasis on tools that support or automate aspects of SCM. It introduces the basic concepts and terminology, describes the areas that are amenable to automation, discusses a representative set of existing SCM tools, and identifies areas for future research and development.

Instructor: Professor Walter Tichy, University of Karlsruhe, Federal Republic of Germany

Tutorial 4: Software Intellectual Property Rights

Intellectual property rights have become an important source of opportunities, as well as an important set of constraints, for the work of software engineers. The objective of this tutorial is to give software engineers a basic understanding of the forms of intellectual property protection available for software and to identify areas where intellectual property law is settled and areas where the law is unsettled as to its application to software.

Instructor: Professor Pamela Samuelson, University of Pittsburgh

Tutorial 5: Software Risk Management

Risk management (RM) is an emerging discipline which provides techniques both for risk assessment (risk identification, analysis, and prioritization) and risk control (risk management planning, resolution, and monitoring). RM techniques have helped many software projects avoid disasters. Up to now, however, software RM techniques have been applied in a largely ad-hoc fashion. This tutorial provides a first cut at establishing a software risk management discipline and will present key principles and concepts of RM, practical techniques for software risk assessment and control, relevant examples of successful risk management approaches and results, and experience in risk assessment on a software project case study.

Instructor: Dr. Barry Boehm, TRW Defense Systems

SPOUSES TOURS

Creative Conventions Services, of Pittsburgh, will be offering a variety of spouses tours during the day on May 15-18. Complete information on the spouses tours and registration materials is included in the Advance Program and will also be mailed to all registrants. Register early as all tours are offered on a space-available basis.

TRAVEL DISCOUNTS FOR ATTENDEES

Domestic & International

Special air-fare prices have been arranged by MON VALLEY TRAVEL of Pittsburgh. For details call 24 hours a day at (800) 245-1099 in the Continental USA and Canada or at (800) 452-2214 within PA. MON VALLEY TRAVEL can also be contacted via facsimile at (412) 765-2614 or telex at 275287.

For a copy of the full Advance Program, contact the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, D.C., 20036-1903 (202) 371-1013.

TOOLS FAIR • May 16-18, 1989

The Tools Fair runs parallel to the Technical Program.

Tools Exhibition

The exhibition will include both software engineering vendors and research organizations. The wide array of software products displayed will allow software professionals to gain first-hand experience in tool development work within the software engineering field. Organizations currently scheduled to exhibit include:

- | | | | |
|------------------------------|-------------------------------|--------------------------------|--------------------------|
| • Cadre Technologies, Inc. | • Jackson Systems Corporation | • Rational | • Tuval Technologies |
| • Carnegie Mellon University | • Knowledge Ware, Inc. | • Ready Systems Corporation | • University of Hawaii |
| • Deft, Inc. | • McCabe and Associates | • Scandura Intelligent Systems | • Verilog |
| • Kokusai Denshin Denwa Co. | • Osaka University | • Systematica Ltd. | • Wayne State University |
| • i-Logix | • ParcPlace Systems | • Syscorp International | |

To request exhibition space, contact the Tools Fair Chair, Grady Booch, at (303) 986-2405 or by facsimile at (303) 987-2141.

Tools Presentation Track

This track is devoted to providing an in-depth description and demonstration of particularly interesting or innovative software tools. Organizations currently scheduled for this track include:

- | | | | |
|-------------------------------|-------------------------|--------------------------------|--------------------------|
| • Cadre Technologies, Inc. | • Loyola College | • Ready Systems Corporation | • University of Hawaii |
| • Carnegie Mellon University | • McCabe and Associates | • Scandura Intelligent Systems | • University of Texas |
| • Hong Kong Polytechnic | • Osaka University | • Syscorp International | • University of Virginia |
| • Jackson Systems Corporation | • ParcPlace Systems | • Tuval Technologies | • Verilog |
| • K&G Automated Systems | • Purdue University | • University of Colorado | • Wayne State University |
| • Kokusai Denshin Denwa Co. | • Rational | | |

TECHNICAL PROGRAM • May 16-18, 1989

The technical program features invited speakers, presentations of refereed papers, and panel discussions. Four sessions will be held on each of the three days, May 16-18, 1989. Each day, a plenary session will feature one or more well-known invited speakers. The remaining three sessions per day will be split into two tracks of panels and paper presentations.

Thirty-six refereed papers will address topics in the following areas:

- | | | |
|----------------------------|----------------------|------------------------|
| • Programming Environments | • Project Management | • Language Issues |
| • Software Metrics | • Methodology | • Analysis and Testing |
| • Specifications | • Process Models | • Software Tools |
| • Software Reuse | | |

Seven panel sessions will cover the following topics:

- | | |
|--|---|
| • A Twenty Year Retrospective of the NATO Software Engineering Conferences | • Modeling of the Software Process |
| • Software Engineering in the Year 2001 | • Complete Validation of Software |
| • Software Engineering Research Agendas | • Report on the Software CAD Databases Workshop |
| • Software Engineering for Business Applications | |

In addition, a software tools track will be conducted in parallel with the three day technical program.

For a copy of the full Advance Program, contact the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, D.C. 20036-1903, (202) 371-1013.

PROGRAM CO-CHAIRS

Richard Fairley
George Mason University, USA
Dines Bjorner
Dansk Datamatik Center, Denmark

PROGRAM COMMITTEE

R. Adrion
K. Agusa
R. Balzer
D. Barstow
V. Basili
F. Belz
D. Berry
M. Boasson
B. Boehm
J. Brackett
S. Chan
L. Clark
R. Conradi
W. Curtis
M. Dowson
T. Denvir
L. Duvall
P. Fowler

P. Freeman
K. Futatsugi
C. Ghezzi
H. Hagelstein
W. Howden
G. Kaiser
T. Katayama
R. Kemmerer
K. Kishida

R. Kurki-Suonio
B. Lang
J. Lavi
M. Lehman
T. Matsubara
M. Moriconi
W. Murray
J. Musa
K. Nori

D. Notkin
P. Oberdorf
M. Ohba
L. Osterweil
M. Penedo
D. Perry
K. Reed
W. Riddle
D. Rombach

R. Selby
A. Sernadas
L. Thorelli
A. Wasserman
J. Wileden
L. Williams
P. Zave
S. Zeil
M. Zelkowitz

CONFERENCE REGISTRATION FORM

1. Send this form to: ICSE-11 Registration
IEEE Computer Society
1730 Massachusetts Avenue, NW
Washington, DC 20036-1903
(202) 371-1013
or submit by FAX to: (202) 728-9614
2. To guarantee your registration, complete the charge authorization or enclose a check payable to IEEE Computer Society.
3. The conference fee includes admission to the technical sessions, admission to the Tools Fair, one copy of the proceedings (except students), refreshments at breaks, and the banquet.
4. The tutorial fee includes tutorial notes and lunch.
5. Members of sponsoring organizations are eligible for reduced rates on both the conference and tutorial fees. Please indicate the sponsoring organization and your membership number when claiming these fees.
6. Refunds (less a \$15 handling fee) will be made if a request is received in writing by May 1, 1989.

(Please type or print)

Name: _____

Organization: _____

Address: _____

Phone: _____

Fax: _____

Tutorial Selection (choose one):

- ☐ Tutorial 1: Formal Methods in Software Engineering
- ☐ Tutorial 2: Making Software Engineering Happen
- ☐ Tutorial 3: Software Configuration Management
- ☐ Tutorial 4: Software Intellectual Property Rights
- ☐ Tutorial 5: Software Risk Management

Advance Registration Fee (before May 1, 1989):

	Member	Non-Member	Student
Conference	\$210	\$265	\$50
Tutorials	\$140	\$180	\$140

Late Registration Fee (beginning May 1, 1989):

	Member	Non-Member	Student
Conference	\$250	\$315	\$60
Tutorials	\$175	\$225	\$175

Banquet: \$30 per guest

Diet Restrictions _____

Payment enclosed: Conference Fee: \$ _____

Tutorial Fee: \$ _____

Banquet: \$ _____

Total: \$ _____

Sponsoring organization: _____

Membership number: _____

Credit Card: ☐ American Express ☐ MasterCard ☐ Visa

Credit Card #: _____

Expiration: _____

Cardholder's name: _____

Signature: _____

A diskette with information about Pittsburgh's technological base will be provided to attendees. Please indicate which format you prefer:

☐ IBM compatible 5 1/4" ☐ Macintosh 3 1/2" ☐ No Preference

HOTEL RESERVATION FORM

1. Send this form to: ICSE11 Housing Bureau
Greater Pittsburgh Convention
& Visitors Bureau
Four Gateway Center, Suite 514
Pittsburgh, PA 15222
2. Please submit your request on this official form. Submit one form per room. If additional forms are needed, photocopy this form or request additional forms from IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, D.C., 20036.
3. All reservations must be in writing and processed through the Housing Bureau. Telephone reservations will not be accepted.
4. Indicate 1st, 2nd, and 3rd choice of hotel and indicate preferred accommodations. A limited number of rooms has been blocked at each of the hotels at the convention rate. Room assignment will be made in order of receipt of official hotel reservation form.
5. Reservation confirmations will be sent directly from the hotels. Rooms will not be held after 6 p.m. unless guaranteed by a credit card or first night's deposit.
6. **Unassigned rooms in the official hotel blocks will be released April 21, 1989.** Beginning April 21, 1989, please make reservations directly with the hotel.
7. Please submit cancellations and changes to the Housing Bureau in writing until April 21, 1989. Beginning April 21, please contact hotel directly.
8. All rates are in U.S. dollars.

(Please type or print)

Name: _____

Organization: _____

Address: _____

Phone: _____

Room preference: ☐ Single (1 bed/1 person)
☐ Double (1 bed/2 persons)
☐ Twin (2 beds/2 persons)

Sharing room with: _____

Arrival: _____ Departure: _____

Hotel:	Single:	Double:	Twin:
☐ Vista	\$83	\$93	\$93
☐ Hilton	\$75	\$85	\$85
☐ Hyatt	\$78	\$88	\$88
☐ Sheraton	\$88	\$98	\$98

Guaranteed by:

Check (payable to ICSE Housing Bureau): Yes ☐ No ☐

Credit card: Yes ☐ No ☐

Credit card: ☐ American Express ☐ Visa
☐ Master Card ☐ Diner's Club

Credit card #: _____

Expiration: _____

Cardholder's name: _____

Signature: _____

Code Optimization Across Procedures

Stephen Richardson and Mahadevan Ganapathi
Stanford University

Code optimization continues to fascinate compiler and architecture researchers. The software community shows increased interest in speeding up the performance of compilers, while computer architects find that trade-offs between hardware and software depend significantly on optimizing compilers. Modern architectures shift more function into the compiler and require a tight integration of compiler technology with the machine. Compiler designers have responded to this need by participating in the design of new architectures. Recently designed architectures promote the advance of new compiler technology and pose challenging optimization problems for compilers, such as pipeline scheduling and parallelism extraction.

In response to the constant pressure for innovation in compiler technology, a number of promising new optimization techniques have been proposed. *Interprocedural dataflow analysis* tracks data across procedure calls. *Profiling techniques* enhance the compiler's ability to accurately predict the probabilistic behavior of program flow. *Procedure integration* and the related technique of *linkage tailoring* allow custom modification of procedures specific to call sites. *Interprocedural register allocation* provides new ideas for designing and using large register files. *Pointer* and *alias tracking* permit even more accu-

Procedure calls can be a major obstacle to the analysis of computer programs, preventing significant improvements in program speed. Can this obstacle be overcome?

rate analysis of dataflow, while *interprocedural constant propagation* has the potential to speed execution by observing the use of constants across procedure boundaries.

Interprocedural analysis techniques range from simple procedure integration to aggressive dataflow analysis.¹ In all these techniques, the major thrust is to improve code optimization using knowledge of the caller and the callee together.

In the calling scenario of Figure 1, information about the caller *P* is potentially

useful in optimization of the body of callee *Q* and vice versa. Some techniques, such as interprocedural constant propagation and alias analysis considering stores to reference parameters, are useful in disseminating caller information to the callee body. Other techniques, such as interprocedural register allocation and procedure integration, are useful in exposing callee information to the caller. These techniques usually target one of the two programming profiles of Figure 2 to perform sophisticated loop optimizations, software pipelining, and code scheduling, and to expose pipeline behavior.

For example, when a merged call occurs within a loop, loop-invariant computations and strength-reduction candidates in the callee can be beneficially moved outside the loop in the caller.

Sorting out the relative efficacies of a variety of powerful optimization techniques requires careful statistical analysis over a range of computer programs. This article concentrates on statistical data relevant to understanding the strengths and weaknesses of proposed new algorithms and heuristics, some of which have been completely implemented and evaluated. It focuses on the interactions between procedure calls and data and also the aliasing patterns that can result from such interactions.

For example, some questions of interest

to the code-optimization community would be: Where in the call graph does a procedure spend most of its time? How often do procedure calls interfere with global dataflow and in what manner? Do variables tend to be used across procedure calls? How often does the use of pointers or pass-by-reference parameters result in potential aliases with other variables? Answers to such questions provide better insight for directing compiler and architecture research.

We place major emphasis on questioning the broad role of aggressive dataflow analysis in code optimization. Statistics motivate the study of interprocedural behavior and reinforce conclusions regarding high-level optimization strategies such as procedure integration, interprocedural register allocation, register windowing, and dependency analysis.

A few words on terminology are in order. *Flow-insensitive interprocedural dataflow analysis* refers to the analysis of dataflow events that might happen on at least one path of execution through a given procedure. Flow-insensitive information is often referred to as *may* summary information because it is information about events that may or may not occur when a procedure is executed. On the other hand, *flow-sensitive interprocedural dataflow analysis* attempts to discover events that must occur on all possible paths of execution through the procedure; this information is referred to as *must* summary information. The term *interprocedural analysis* in the literature usually refers to flow-insensitive interprocedural dataflow analysis. Similarly, references to *interprocedural optimization* usually refer to code optimization based on flow-insensitive interprocedural dataflow information.

The following sections consider a broad range of techniques, each of which is in some sense interprocedural by nature. Some techniques rely on interprocedural dataflow in their analysis. Other techniques require interprocedural information in the form of detailed profile data or information concerning the scope of a given procedure in relation to other procedures. All these techniques are admissible under the umbrella term interprocedural analysis.

We reinforce the discussion on interprocedural techniques with statistics and examples from large real programs. Each benchmark has between 250 and 14,000 lines of code and is composed of at least 10 procedures. The average benchmark has 2,673 lines of code and 65 procedures. The benchmarks are described in more detail in Table 1 and in the sidebar.

Interprocedural dataflow analysis

Interprocedural dataflow analysis has been actively researched for almost two decades,¹ with the major emphasis on efficiently computing flow-insensitive Mod (modify) and Use sets for each procedure in a program. If a variable is a member of the Mod set for a given procedure, then it might be modified by execution of that procedure. Similarly, a variable that is a member of the Use set of a procedure might be used during execution of that procedure.

Interprocedural dataflow analysis solves dataflow in a program at a procedure-call site. For example, consider a program that uses variable *i* both before and after a call to procedure *P* (see Figure 3). If *i* is global to *P*, then there is a possibility that *i* is modified or used by *P*. Aggressive interprocedural dataflow analysis determines whether *i* is a member of Mod(*P*) or Use(*P*). As noted by many researchers, different types of code-optimization techniques can benefit from answers to such interprocedural questions:

- enhancement of standard intraprocedural optimizations such as common subexpression elimination, redundant code elimination, strength reduction, and induction variable elimination;
- interprocedural register allocation;
- procedure integration and linkage tailoring;
- interprocedural constant propagation; and
- dependency analysis for parallelism and vectorization.

In addition, interprocedural dataflow analysis has been proposed for a number of diverse applications. Program verifiers need interprocedural dataflow analysis to build cross-references and connect variable definitions to possible uses.² Similarly, interprocedural dataflow information helps program editors identify parallelism and is potentially useful in automatic dependency analysis and code optimization.^{3,4}

A variable is live at a point in the program if it has a potential use preceding stores into it on a subsequent path. A live range of a variable is a section of the program during which a given definition of that variable is in use. Subsequently, the variable might not be in use any longer or a new definition for that variable might be enforced.

One factor in determining the utility of interprocedural dataflow analysis for se-

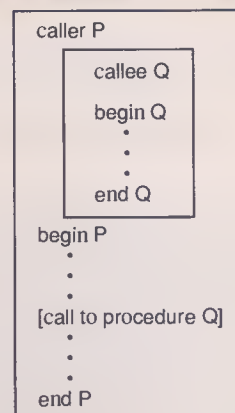


Figure 1. *P* calls *Q*.

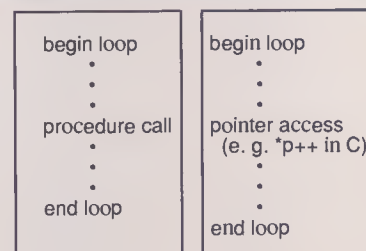


Figure 2. Program profiles targeted by interprocedural techniques.

Table 1. Benchmark statistics.

Filename	Lines	Procedures
bench	941	48
bigfm	541	18
ccal	799	27
dhrystone	251	12
dnf	1,585	37
hopt	2,197	92
macro	7,484	179
newtlb	355	14
pwhets	350	10
simu	3,458	101
tlb	354	114
upasmips	13,768	235
(average)	2,673	65

```

i := 3;
call P;
y := a[i]; —(is the value of i equal to 3 after the call?)

```

Figure 3. Interprocedural dataflow: is i equal to 3 after the call to P ?

Benchmarks

The benchmarks used in the evaluation of interprocedural techniques are

bench	Combination of the 10 Stanford benchmarks: bubble, dnf, perm, queen, tower, quick, tree, mm, puzzle, and fft.
bigfm	An implementation of the Fiduccia-Mattheyses algorithm as described in "A Linear Time Heuristic for Improving Network Partition," <i>Proc. 19th Design Automation Conf.</i> , June 1982, pp. 175-181. This algorithm reads a netlist of a hypergraph from standard input and produces a partition of that graph into approximately equal sections, written to standard output. The algorithm adapts the Kernighan-Lin algorithm for graph partition by adding data structures to speed the calculation of the biggest gain and the updating of gains of neighbors for each move.
ccal	Interactive desk calculator, written by Warren Cory of Stanford University.
dhystone	A synthetic benchmark developed by Reinhold Weicker.
dnf	A normalizer from arbitrary Boolean expressions to disjunctive normal form.
hopt	Optimizer for intermediate code, written by Edwin Hou of Stanford.
macro	Macro expansion preprocessor for the SCALD (Structured Computer-Aided Logic Design) computer-aided design system developed by T.M. McWilliams and L.C. Widdoes of Stanford University. Described in technical report 152, Digital Systems Laboratory, Stanford University, March 1978.
newtlb	Translation look-aside buffer and cache simulator.
pwhets	A Pascal version of the whetstone program.
simu	Simulation of an operating system.
tlb	Another translation look-aside buffer and cache simulator.
upasmips	A Pascal compiler for the Stanford U-Code system.

quential code optimization is the degree to which live ranges of global variables are actually interrupted by procedure calls. This factor, termed $LiveInt_{call}$, is calculated as follows¹:

(1) If the live range of a variable v includes a call to a procedure P such that v is available to the scope of P , the integer variable i_v is valued one. (For simplicity, we have assumed that if a given variable is live anywhere within a procedure, it is live at each call site within the procedure.)

(2) The program is run and a dynamic count, n_v , is taken of the number of uses of each variable v .

(3) Interference $LiveInt_{call}$ is calculated as follows:

$$LiveInt_{call} = \sum n_v i_v / \sum n_v$$

$LiveInt_{call}$ for each of a number of representative Pascal benchmarks appears in Table 2. Without interprocedural optimization, procedure calls interfere with up to 22 percent of the dynamic loads and stores in a program. For each variable n_v whose live range intersects a procedure call, interprocedural analysis determines whether the procedure call actually interferes with that variable. If no interference exists, the vari-

able is subject to optimization.

The amount of speedup gained by the optimization depends on how often the variable is used dynamically. The $LiveInt_{call}$ figures imply that if interprocedural dataflow analysis were to find that procedure calls interfere with none of the variables n_v , a program would run about 22 percent faster on average. In reality, many of the variables n_v actually are modified by one or more procedure calls in their live ranges, and the program speeds up much less than 22 percent. We calculated interprocedural information and entered it in a global code optimizer for Pascal to enhance existing optimizations, including global common subexpression elimination, redundant code elimination, strength reduction, and induction variable elimination. As shown in Table 2, the actual speedup averages 1.57 percent.

Therefore, procedure calls seldom interfere with variables that are global to the called procedure. The live ranges of about 78 percent of such variables do not span procedure calls. Furthermore, the 22 percent remaining variables whose live ranges do span procedure calls generally have their values changed by the callee, a fact that is conservatively assumed true by the compiler in the absence of interprocedural information. Thus, the additional benefit of interprocedural dataflow analysis for sequential code optimization becomes an insignificant 1.57 percent.

Triolet proposed using interprocedural analysis to parallelize code via automatic dependency analysis,⁴ whereas attempts to perform optimization of sequential code have been somewhat disappointing.¹ This dichotomy exists because of potentially higher stakes in parallel applications. For example, identifying nondependencies within a loop can allow that entire loop to speed up by a factor equal to the number of loop iterations. However, identification of global common subexpressions or redundant memory accesses as a result of interprocedural dataflow analysis might only save a few cycles per loop iteration. These savings are usually small when compared to the running time of the remainder of the loop. With the exception of alias information being potentially useful in leaf procedures, interprocedural dataflow analysis cannot speed up code within a leaf procedure. (A leaf procedure is one that calls no other procedures. A program can be thought of as a tree whose root is the main procedure and whose branches represent various paths of execution through other procedures. A leaf procedure terminates each branch.)

Further discussions regarding these issues are relegated to a subsequent section on dependence analysis.

Procedure integration

Selective integration of procedures provides an attractive alternative to interprocedural dataflow analysis. In procedure integration, also known as in-lining or merging, the body of a given procedure is copied in place of the procedure call. This process is essentially equivalent to implementation of call-by-name semantics of procedure calls. When a procedure is merged, straightforward intraprocedural techniques, as in intraprocedural global code optimization, are sufficient to track dataflow.

Register allocation can be immediately seen as more efficient. The register set must be large enough so that a completely nonoverlapping set of registers can be allocated to local variables within the merged procedure. If a global value is used both outside and inside a merged procedure, it can reside in the same register during its lifetime. Without merging, extra save and restore commands would be necessary. In addition, on many architectures, procedure call and return instructions are expensive. In these cases, significant execution time savings can result from procedure integration.

The major disadvantages of procedure integration are the rapid increase in code size and substantial increase in time needed to perform the integration. Furthermore, procedure integration is an incomplete solution when considering recursive procedures and memory space limitations imposed by current software and hardware technologies. More memory space is required to store relocatable and executable forms of a program. Execution speed can also be adversely affected by the increased instruction cache miss rate. We should mention that, for this operation, increased cache miss would be more typical than decreased cache miss.

If the code increase necessitated by complete in-line expansion is unacceptably large, the compiler must selectively merge only those procedures expected to yield substantial benefits. Optimally, of course, we would merge only those procedures in which the program is likely to spend the most time. These procedures could be identified by profiling or any one of a number of heuristic techniques. One technique that might be particularly effective is to merge only leaf

Table 2. Removable interference due to procedure calls.

Filename	<i>LiveInt_{call}</i>	Speedup
bench	3 %	-0.365 %
bigfm	11 %	-0.334 %
ccal	18 %	1.075 %
dhystone	32 %	0.597 %
dnf	1 %	0.000 %
hopt	12 %	0.004 %
macro	26 %	-0.066 %
newtlb	27 %	7.026 %
pwhets	19 %	0.322 %
simu	13 %	0.192 %
tlb	59 %	10.723 %
upasmips	39 %	0.004 %
(average)	22 %	1.571 %

Table 3. Time spent in leaf procedures.

Benchmark	Leaf+Sys	Leaf/User	Leaf/Total
bench	99.33 %	16.67 %	0.13 %
bigfm	94.63 %	55.06 %	6.58 %
ccal	65.33 %	11.22 %	4.38 %
dhystone	87.41 %	74.93 %	37.63 %
dnf	65.71 %	46.43 %	29.71 %
hopt	90.66 %	68.21 %	20.04 %
macro	66.34 %	26.81 %	12.33 %
newtlb	72.66 %	42.55 %	20.24 %
pwhets	83.08 %	75.60 %	52.41 %
simu	61.03 %	19.92 %	9.69 %
tlb	47.93 %	44.35 %	41.49 %
upasmips	40.88 %	26.62 %	21.44 %
(average)	72.92 %	42.36 %	21.34 %

procedures, since it has been claimed that a program spends much time at the leaf level.⁵

Table 3 shows what percentage of time is spent in the procedure leaves for a number of benchmarks. In this table, system calls and library functions are not counted as procedures; that is, if a procedure calls only standard library routines, it still counts as a leaf procedure. The user time of a program is defined as that portion of its time spent in user space, that is, while not executing system calls and library functions. The first column of the table shows what percentage of execution time is spent executing leaf procedures and system calls. The second

column shows what percentage of user time is spent executing leaf procedures, and the third column shows what percentage of total execution time is spent executing only leaf procedures.

Examination of Table 3 reveals that, by merging system calls and library functions, code responsible for 72.92 percent of execution time is exposed for further optimization. Leaf procedures alone account for 42.36 percent of the time spent executing the user part of the program and 21.34 percent of the total execution time.

Of course, other compile-time methods of limiting in-line expansion are possible.⁶ Among the decision-making strategies that

Table 4. Comparison of register allocation schemes.

	Global Coloring	Link-time Allocation	Compile-time Allocation
Local coloring?	yes	yes	yes
Global coloring?	yes	no	no
Usage estimates at variable level?	yes	yes	no
Usage estimates at procedure level?	no	no	optional
Interprocedural dataflow analysis required?	yes	no	no

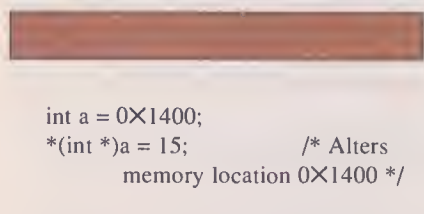


Figure 4. The aliasing problem in C.

can be employed are

- merging only small procedures,
- merging procedures based on runtime statistics,
- predicting interprocedural constants with knowledge of constant valued parameters,
- merging only certain calls as deemed important by profiled execution, and
- merging calls within loops only.

A related optimization technique is to *tailor* the linkage code at each call site.³ Linkage tailoring has often been cited as a less memory-expensive alternative to procedure integration. In linkage tailoring, one or more copies of a procedure are made and each version is streamlined for a given class of procedure calls. For instance, suppose procedure *P* is called half the time with the constant parameter "true" and half the time with the constant parameter "false." Two copies of *P* are made: *P*_{true} and *P*_{false}, each tailored to fit one class of procedure call. Previous calls to *P*(true) and *P*(false) are replaced by calls to *P*_{true} and *P*_{false}, respectively.

Effective linkage tailoring requires extensive analysis similar to interprocedural dataflow analysis. All calls to a given procedure must be examined for possible patterns in the parameter list. Subsequently, a decision-making algorithm must determine whether the procedure is called often enough to make tailoring worthwhile. In contrast, it is indeed true that simple in-line expansion can provide excellent speedup with a reasonably small amount of code

expansion. We must consider these factors when evaluating linkage tailoring as a suitable alternative to procedure integration.

Interprocedural constant propagation can be considered a subset of the linkage tailoring technique. In this optimization technique, all calls to a given procedure are carefully analyzed to determine the existence of parameters whose values are constantly the same for all similar call sites in the program. When such cases are identified, procedures carrying these constant parameters can be optimized.

Cooper et al. described an efficient procedure for computing interprocedural constants in a programming environment, in which constants are to be propagated before parallelism detection.³ They indicated that this technique will approach the effectiveness of in-line substitution. However, experimental results have yet to confirm this notion.

Finally, it is important to examine another strategy that has been implemented and is gaining acceptance as a useful alternative: interprocedural register allocation.

Interprocedural register allocation

Interprocedural register allocation is an optimization technique for efficiently using large register files. Often procedures have few local variables and small basic block sizes. Consequently, reasonably good intraprocedural register allocators seldom need more than six registers. By considering program behavior across procedure boundaries, we can increase the size of the variable pool along with the size of effective code blocks.

Graph coloring has been proposed in various forms and subsequently refined, extended, and implemented for register allocation within individual procedures.⁷ The coloring algorithm constructs an interference graph whose nodes represent the live ranges of variables in the program. The

edges in the graph represent overlap or interference between live ranges. If *r* is the register count considered for register allocation, the graph is colored with *r* colors such that no two adjacent nodes share the same color. This graph coloring problem is known to be NP-complete. When the graph cannot be colored, values are spilled to memory to reduce the number of colors in the graph.

Interprocedural register allocation schemes differ mainly in their degrees of aggressive coloring:

- coloring an entire program,
- coloring on a procedure basis with knowledge of the program's call graph, and
- extending intraprocedural coloring to minimize register use penalty at call sites.

Perhaps the most elaborate and complete method of interprocedural register allocation would use interprocedural dataflow analysis in conjunction with a program's call graph to fully color all variables and constants in the program. This proposal envisions register allocation as optimal with respect to spills. Minimization of register save and restore costs requires interprocedural analysis to select optimal calling conventions on a call-site basis. Furthermore, some degree of pointer and alias analysis is essential to allow more variables to reside in registers.

Implementing such a scheme at compilation time to analyze programs is quite expensive and the potential benefit, unknown. However, we can obtain an indication of its benefits by considering existing implementations of subsets of the complete scheme.

Chow discussed an extension of a priority-based graph-coloring algorithm to obtain the benefits of interprocedural register allocation.⁸ Procedures are processed in depth-first ordering of the call graph. Register usage penalty at call sites is effectively reduced by making good use of a large number of registers.

Wall considered interprocedural register allocation at link time.⁹ In this scheme, interprocedural coloring of globals is not performed, although locals are colored. Pointer tracking is not considered; a reference to the address of a variable is rendered ineligible for allocation. At link time, each eligible variable is placed into a group of variables whose live ranges do not overlap. Worst-case estimates are used to determine live ranges: locals are assumed live during the entire lifetime of the procedure in which

they are defined, and globals are assumed live throughout the entire duration of the program. Registers are assigned to variable groups according to their frequency of use. This frequency is determined either by profiling or by using simple heuristics.

Results obtained using this scheme are certainly encouraging. Using 52 registers, an average speedup of 14 percent is obtained on six benchmarks. Contrast this result with 9 percent for compile-time allocation without coloring and 10 percent for compile-time allocation with coloring.⁹ Optionally turning on local coloring intra-procedurally does not significantly alter these results.

Steenkiste and Hennessy employed a simplification of the above technique when considering interprocedural register allocation for Lisp on reduced instruction set computers.⁵ The locals of each procedure are colored independently. Variables are then allocated to registers in a bottom-up fashion: variables in leaf procedures are allocated first, then variables in the next level outward. Optionally, profiling can be used to estimate whether it is worthwhile to perform register allocation on a given procedure. This scheme has the advantage that it can be performed at compile time, eliminating the need to rewrite code while linking modules together. Furthermore, its simplicity obviates an intermediate step in the register allocation process where variables need to be assigned to pseudo-registers.

Table 4 summarizes the different schemes: interprocedural register allocation by a full global coloring of the entire program; Wall's link-time allocation scheme; and Steenkiste and Hennessy's compile-time scheme.

Pointer and alias tracking

Dataflow analysis attempts to precisely identify the definitions and uses of each variable in a program: where its value is first defined, where the value is redefined (killed), the portion of the program that uses the variable (live range), and other relevant information potentially useful in code optimization. A crippling obstacle to flow analysis is aliasing, the possibility that one memory location can have two or more different names. Obviously, if variables *a* and *b* point to the same memory location, *a* is killed when *b* is killed and vice versa. Unfortunately, aliasing patterns are seldom simple; disastrous consequences might result depending on the freedom allowed

Table 5. Removing interference due to aliasing.

Filename	Potential	Actual
bench	0.0 %	0.000 %
bigfm	0.3 %	0.000 %
ccal	3.6 %	-0.015 %
dhystone	4.1 %	0.397 %
dnf	8.8 %	0.002 %
hopt	11.8 %	0.711 %
macro	10.2 %	0.003 %
newtlb	0.0 %	0.000 %
pwhets	3.4 %	-0.146 %
simu	5.3 %	0.082 %
tlb	0.0 %	0.000 %
upasmips	16.2 %	0.022 %
(average)	5.3 %	0.088 %

by programming language semantics. Uncertainty as to whether a given variable is aliased can often invalidate the dataflow information calculated for that variable.

In the C language, the aliasing problem is severe; any given variable can be treated as a pointer and the memory space pointed to by the variable is unrestricted (see Figure 4). Thus, any access via a pointer of unknown value must necessarily be treated as a potential access to all memory locations. However, the frequency of dynamic pointers in C is often small and thus the impact of a worst-case assumption on execution performance need not be large.

In Fortran, aliasing can arise from the use of either common blocks or procedure arguments; all arguments are passed by reference in Fortran. Common block aliasing is a variety of renaming. The possibility that the same actual can be passed to two formals is more insidious. However, it is unlikely that this form of aliasing is a major problem in practice. While not providing any quantitative numbers, previous research indicates that aliasing is indeed rare in Fortran programs.¹⁰

In Pascal, aliasing arises under restricted circumstances. For example, a global variable can be passed to a procedure as a Var parameter. That is, the called procedure receives a pointer to the variable's location in memory rather than receiving a copy of the contents of the variable. In this case, the global and the parameter will both refer to the same memory location. As in Fortran, the same actual might be bound to two different reference formal parameters. Again, it is questionable how often such

aliasing occurs in real programs.

Experiments with Pascal programs indicate that the amount of interference in the dataflow graph caused by potential aliasing is indeed small. Table 5 presents the potential interference possible in various programs based on statistical analysis of dataflow characteristics from a dynamic trace of each program and compares this theoretical upper bound with results found by implementing an alias analysis package within an optimizing compiler.¹ The actual speedup reported is the incremental speedup found by comparing a program optimized in the presence of alias information against a program optimized without such information. Once again, only a small percentage of the original interferences can be removed using expensive, precise information. In this case, the average benchmark had 5.3 percent of its loads and stores interfered with in some manner by simple aliasing. Of this 5.3 percent, the optimizer managed to remove only 0.088 percent with additional alias information.

The small negative number associated with some benchmarks reflects a certain amount of random behavior on the part of the compiler. Very small changes in state, such as the addition of negligible amounts of alias information, can cause tiny unpredictable changes in the running time of the program.

To the extent that other languages have aliasing patterns similar to Pascal, the result of performing this experiment with compilers for other languages will be similar. As noted previously, aliasing in Fortran occurs in much the same manner as Pascal. Because the scoping structure for Fortran is even simpler, the overall effect might be that very little aliasing occurs. On the other hand, aliasing in C is much more likely due to the ease with which aliases form in this language. We expect somewhat more aliasing; exactly how much more we don't know at present.

Dependency analysis

Procedure calls can impede automatic identification of parallelism in computer programs. Several researchers have proposed the use of interprocedural dataflow information for restructuring sequential programs to run on parallel machines.⁴ The potential benefits of such information have yet to be ascertained. Although interprocedural dataflow information has produced disappointing results when applied to conventional optimizations on sequential machines, sophisticated interactions be-

```

for i := 1 to 50 do
  a[i] := a[i] + g;
  call Q;
end for;

```

Figure 5. Ten cycles of computation per loop; g is global.

tween complex optimizations in a parallel processing environment might nevertheless profit from interprocedural analysis.¹¹

During our earlier discussion on interprocedural dataflow analysis, the figure of merit $LiveInt_{call}$ was used to predict a maximum bound on program speedup obtainable by applying interprocedural information to a code optimizer. On a parallel machine, the scenario can be quite different. Consider the example of Figure 5, depicting a procedure call within a loop.

The call to procedure Q interferes with the load of global variable g . The potential payoff occurs when interprocedural analysis determines that the call to Q does not alter the contents of g . In the sequential case, this analysis permits replacement of the load of g from memory with either a constant load or a register load. This replacement saves perhaps one cycle out of the 10 required to execute the loop, resulting in a speedup of 10 percent. In the parallel case, it is possible that interprocedural information will enable parallelization of the entire loop, resulting in a speedup factor of 50. Other interesting scenarios to consider are loops with vector statements and triangular loops.

The payoff per optimization on parallel machines can exceed that for sequential machines, but discovering specific opportunities for optimization can be quite difficult. In the previous example, sequential optimization benefits by simply observing noninterference between procedure call Q and global variable g . However, if the loop needs parallelization, a stricter requirement than Q not modifying g must be met: Q itself must be loop independent.

Therefore, the benefits of interprocedural information to dependency analysis do not relate to the number of loads and stores that can be eliminated. To evaluate the utility of this technique, we must employ a new metric. Undoubtedly, this metric must consider privatization of variables within loops, minimization of cache miss, special cache strategies such as instruction cache

for leaf procedures, and cache coherency in a shared-bus architecture where it can be less expensive to retrieve a shared variable in another cache than from memory.

On the experimental front, Triolet reported promising results when automatically parallelizing Do loops in the Linpack library with the help of interprocedural dataflow analysis.⁴ Researchers have yet to report quantitative speedup for entire programs subject to parallelization.

Burke and Cytron proposed a dependence test that includes precise alias information by propagating assertions about ranges.¹¹ However, dependency analysis using interprocedural information has not tangibly increased a compiler's ability to vectorize programs.¹⁰ Similarly, vectorization has yet to prove applicable to a significant portion of program space.

Discussion and conclusions

Over the past two decades, researchers have come to thoroughly understand interprocedural dataflow analysis. Simple interprocedural summary sets such as modification and use of variables provide little help in optimization of sequential code. Sophisticated interprocedural dataflow analysis focuses on parallelizing loop iterations and includes dependence analysis at the array-element level across procedure boundaries. Recently, algorithms have been developed for interprocedural subscript analysis to determine independence at the array-reference level in the presence of interprocedural aliasing. However, we lack conclusive evidence for their usefulness in code optimization when applied to vectorization and parallelization.

Interprocedural experiments on sequential code reveal almost no interference between procedure calls and global dataflow. Some researchers have found interprocedural Mod and Use information so broad as to not usefully refine intraprocedural information. Others have determined that compile-time approximations to runtime events are adequate; interprocedural Mod and Use information is sharp, but no optimization opportunities exist to use this additional information and thus impact runtime.

Alias information turns out to be least useful. Pascal offers certain difficulties in detecting aliases: between Var parameters and global variables, Var parameters and heap objects, Var arrays and Var scalars, and with-accessed parts of structures and directly accessed counterparts. In a strongly typed language such as Pascal, a

variable of one type cannot alias an object of another type. Similarly, heap objects cannot be aliased with nonheap objects. Usually, variables are passed by value rather than by reference, in which case aliasing is impossible. However, each access to a Var parameter can potentially kill globals held in registers and vice versa. Experiments confirm that alias patterns are not complex.

Perhaps block-structured programming languages inhibit benefits that some other programming paradigm would not. Languages such as Pascal encourage the programmer to declare good approximate information for the compiler. For example, explicit differentiation between call-by-reference and call-by-value parameter binding narrows the distance between a compiler's worst-case estimate of a Mod set and the precise information derived from flow-insensitive summary analysis. Often, in languages such as Ada, observing language visibility rules obviates summary analysis. In contrast, in a language like Fortran where all parameters are call-by-reference, the worst case estimate can differ strikingly from the results of summary analysis.

At the same time, interprocedural Use information mostly benefits global variables; programmers rarely pass a variable into a procedure unless it is used or modified. Thus, most call-by-value parameters are used in the callee procedure. Since there does not exist a fundamental difference in patterns for global variables in Pascal and Fortran, we expect interprocedural Use information to be unprofitable in the Fortran context as well.

While interprocedural dataflow analysis provides little help in optimizing sequential code when combined with intraprocedural global optimization techniques, the benefits of its use with other optimization techniques (such as interprocedural constant propagation, even of immediate constants) are unclear. Approximate methods have been proposed to evaluate variables that have known constant values on invocation of various procedures in a program. For example, loop upper bounds can make a difference in the results of array independence analysis. Similarly, information about array dimensions and loop index strides and extents can be improved by interprocedural constant propagation, resulting in improved loop optimization. However, no empirical results exist on the usefulness of interprocedural constant propagation and, in particular, its usefulness in dependence testing. In the absence

of evidence, it is indeed questionable whether this technique will approach the effectiveness of procedure integration.

In contrast, interprocedural register allocation techniques have proven effective at reducing the register save/restore traffic at procedure calls, effectively reducing register use penalty at call sites. These methods rely on a large number of available registers and have improved performance when applied at link time or at compile time.

Since the largest potential gains come from enhanced parallelism, recent research has concentrated on compiling for parallelism. Information describing the extent and pattern of both intra- and interprocedural array access information is critical for a parallelizing compiler. Automatic techniques have been proposed to minimize memory overhead either through good register allocation or by regularizing memory accesses using vector operations. The success of a vectorizer is attributable to its ability in analyzing loops and subscripts.

Dependences ultimately occur between references to memory. Dependence analysis is a technique that accounts for subscripts and range information. This analysis is a two-step process: a dependence test is used to formulate a dependence equation, then a decision algorithm determines if the dependence equation has solutions. Structures of different shapes that share storage and dependences hidden by procedure calls force dependence analysis to be embellished with precise aliasing and interprocedural information.

Novel methods of code optimization must influence computer architecture design.¹² From a software viewpoint, an architectural design approach needs to identify the improvement in performance due to any new feature. For example, if a software technique reduces the number of loads and stores in an average program, it alleviates the need for super-fast memory systems in hardware. In a more general vein, if code optimizers work well on simple instructions and poorly on complex instructions, it becomes hard to justify complex instructions in hardware.

In the context of architectural design, the benefit of successful interprocedural optimization is — at least — threefold:

- by moving instructions across procedure calls, a greater amount of useful computation is available to fill deep pipelines, long instruction words, and other delay slots;
- interprocedural register allocation techniques alleviate the need for register windows: special-purpose hard-

ware designed to reduce the overhead cost of register saves and restores at procedure call sites; and

- interprocedural analysis can possibly increase the amount of parallelism in programs, thus potentially influencing the design of larger numbers of processing elements in a multiprocessor system.

On the software side, designers must carefully consider the effect of interprocedural optimizations in the design of every component of a programming environment. Interprocedural summary information helps minimize the costs of recompilation by allowing linkers, compilers, and language-based editors to incrementally update dataflow information.³ Interprocedural information can also be useful for various aspects of software maintenance and debugging.²

The abundance of interprocedural techniques merits close scrutiny of the potential costs and benefits of each scheme. To illustrate the trade-offs

among these techniques, we have underscored their advantages and disadvantages. Wherever possible, we have presented statistical information based on actual implementations.

We expect research focusing on a combination of interprocedural techniques to keep ambitious compiler writers and computer architects busy for the foreseeable future. □

Acknowledgments

This research was supported in part by DARPA-MDA90383C0335 and NSF-DCR8351269. We also wish to thank some of the people whose valuable help and thoughtful input aided the article in so many ways. Our thanks go to Al Aho, Fran Allen, Jeff Barth, Mike Burke, Charles Fischer, John Hennessy, Ken Kennedy, and Steve Tjiang.

FULL AT&T C++: ANNOUNCING VERSION 1.2

Guidelines announces its port of **version 1.2** of AT&T's C++ translator. As an object-oriented language C++ includes: classes, inheritance, member functions, constructors and destructors, data hiding, and data abstraction. "Object-oriented" means that C++ code is more readable, more reliable and more reusable. And that means faster development, easier maintenance, and the ability to handle more complex projects. C++ is **Bell Labs' answer to Ada and Modula 2**. C++ will more than pay for itself in saved development time on your next project.

C++

from **GUIDELINES** for the IBM PC: \$295

Requires IBM PC/XT/AT or compatible with 640K and a hard disk.

Note: C++ is a *translator*, and requires the use of Microsoft C 3.0 or later.

Here is what you get for \$295:

- The full AT&T v1.2 C++ translator.
- Libraries for stream I/O and complex math.
- **The C++ Programming Language**, the definitive 327-page tutorial and description by Bjarne Stroustrup, designer of C++.
- Sample programs written in C++.
- Improved installation guide and documentation.
- 30-day money-back guarantee.

NOW AVAILABLE FOR UNIX V/386 — \$495

To Order:

send check or money order to:

GUIDELINES SOFTWARE, INC.
P. O. Box 749, Dept. CT
Orinda, CA 94563

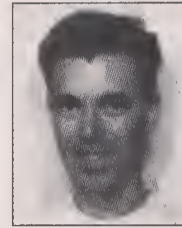
To order with VISA or MC,
phone (415) 254-9183.
(CA residents add 6% tax.)

C++ is ported to the PC by Guidelines under license from AT&T.

Call or write for a free C++ information package.

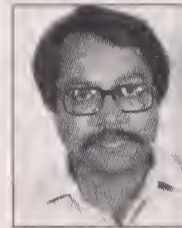
References

1. S. Richardson and M. Ganapathi, "Interprocedural Optimization: Experimental Results," *Software—Practice and Experience*, Vol. 19, 1989 (to appear).
2. B.G. Ryder, "An Application of Static Program Analysis to Software Maintenance," *Proc. 20th Hawaii Int'l Conf. System Sciences*, Jan. 1987, pp. 82-91.
3. K. Cooper, K. Kennedy, and L. Torczon, "The Impact of Interprocedural Analysis and Optimization in the Rⁿ Programming Environment," *ACM Trans. Programming Languages and Systems*, Vol. 8, No. 4, 1986, pp. 491-523.
4. R. Triolet, *Interprocedural Analysis for Program Restructuring with Parafrase*, tech. report 538, Center for Supercomputer Research and Development, Univ. of Illinois, Urbana, 1985.
5. P. Steenkiste and J. Hennessy, "Lisp on a RISC," *Computer*, Vol. 21, No. 7, July 1988, pp. 34-45.
6. F.E. Allen et al., "The Experimental Compiling System," *IBM J. Research and Development*, Vol. 24, No. 6, Nov. 1980, pp. 695-715.
7. F.C. Chow and J.L. Hennessy, "Register Allocation by Priority-Based Coloring," *Proc. SIGPlan Symp. Compiler Construction*, ACM, June 1984, pp. 222-232.
8. F.C. Chow, "Minimizing Register Usage Penalty at Procedure Calls," *Proc. SIGPlan '88 Conf. Programming Language Design and Implementation*, ACM, June 1988, pp. 85-94.
9. D.W. Wall, "Register Windows vs. Register Allocation," *Proc. SIGPlan '88 Conf. Programming Language Design and Implementation*, ACM, June 1988, pp. 67-78.
10. F. Allen et al., *The Parallel Programming Technology Project*, research report, IBM, Yorktown Heights, New York, 1986.
11. M. Burke and R. Cytron, "Interprocedural Dependence Analysis and Parallelization," *Proc. SIGPlan '86 Symp. Compiler Construction*, ACM, June 1986, pp. 162-175.
12. J.L. Hennessy et al., "Hardware/Software Trade-offs for Increased Performance," *Symp. Architectural Support for Programming Languages and Operating Systems*, ACM, March 1982, pp. 2-11.



Stephen Richardson graduated from the University of Tennessee in 1983 with a bachelor's degree in electrical engineering. In 1985 he received an MSEE from Stanford University. Currently, he is working on a PhD program in computer systems at Stanford, where he has been associated with the MIPS-X architecture project.

Richardson's interests are in computer architecture and compiler technology. Most recently, his research has focused on making programs run faster by removing barriers to efficient optimization of compiler-generated code, specifically problems dealing with pointers, aliases, and subroutine calls.



Mahadevan Ganapathi obtained his bachelor's degree in electronics and communication engineering from the Indian Institute of Technology in 1976 and his PhD in computer sciences from the University of Wisconsin at Madison in 1980. He is affiliated with the Computer Systems Laboratory at Stanford University.

Ganapathi's interests are in programming languages, compiler construction, and computer architecture. His specific efforts have focused on the design of intermediate representations, code generation, and code optimization. His additional interests include the theory of parsing and translation, attribute grammars, and attributed parsing. Related research involves retargetable code generation and peephole optimization algorithms. His recent efforts include interprocedural analysis and optimization, and compilers for functional languages.

DEPARTMENT HEAD - COMPUTER SCIENCE University of Lowell Lowell, Massachusetts

The faculty of Computer Science is seeking a new Department Head to provide leadership in research and to administer a comprehensive Graduate and Undergraduate program. This search will continue during the 1988-89 academic year or until the position is filled. The Department Head will be responsible for faculty recruitment and evaluation, development of a comprehensive research plan and the management of all departmental resources. A major priority is to recruit new faculty and expand research activity and the graduate program in Computer Science.

The successful candidate must hold an earned doctorate in Computer Science or a directly related field and must be a citizen or permanent resident of the U.S. The candidate should be nationally recognized and currently active in research and teaching in at least one area of modern Computer Science. Prior experience in academic and research administration sufficient to qualify for a tenured position at the rank of full professor is also required.

For more information about the CS Department, see our prior ad in the January issue of IEEE Computer, ACM Communications or the Affirmative Action Register. Interested applicants are invited to forward a complete academic vitae and names of three references (two in his/her research area), and a letter stating his or her approach toward expanding research and graduate study in Computer Science at the University of Lowell.

Address replies to:

Professor Robert Lechner
Department Head Search Committee
Computer Science, WL229
University of Lowell
Lowell, MA 01854

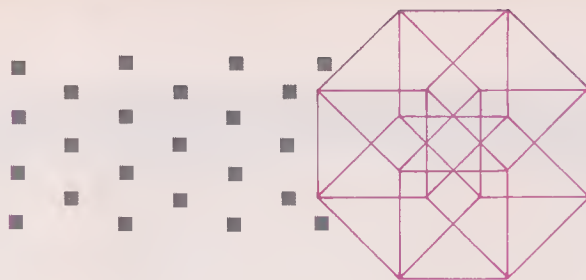
The University of Lowell is an Equal Opportunity/Affirmative Action, Title IX, 504 Employer.

University of
Lowell

Readers may contact Richardson at 25 Center for Integrated Systems, Via Palou, Stanford, CA 94305.

COMPUTER

Fourth Conference on Hypercube Concurrent Computers and Applications



HCCA4

Organizing Committee

Gil Weigand
General Chairman
Sandia National Laboratories

John Gustafson
Program Chairman
Sandia National Laboratories

Bill Hickey
Conference Administration

Program Committee

Don Austin
Department of Energy

Tony Chan
University of California
at Los Angeles

Terry Cole
Jet Propulsion Laboratory

Erik DeBenedictis
AT&T/Bell Labs

Geoffrey Fox
California Institute of Technology

Bill Gropp
Yale University

Michael Heath
Oak Ridge National Laboratory

Paul Messina
California Institute of Technology

Gary Montrey
Sandia National Laboratories

Ed Oliver
Air Force Weapons Lab

Quentin Stout
University of Michigan

HYATT REGENCY CONVENTION CENTER MONTEREY, CALIFORNIA

MARCH 6-8, 1989

Technical Sessions

Concurrent Languages
Performance Evaluation
Scientific Applications
Load Balance/Mapping
Programming Environments
Non-Numerical Algorithms

Architectures
Graphics
Debugging
Graphs/Networks
Optimization
Database Methods

Numerical Algorithms
Operating Systems
Parallelization Tools
Image Processing
Artificial Intelligence
Matrix Methods

Mini-Symposia

Fault Tolerance
MCC Programs in Parallel Software

Neural Networks
Database Applications

Registration

	before 2/15/89	after 2/15/89	
Conference (including Banquet and Proceedings)	\$200	\$250	Make checks payable to: HCCA4.
Conference-Student (Proceedings only)	\$100	\$150	After 2/15/89 registration will be
Additional Banquet	(\$30 per person)		available only at the conference.

☐ Check	☐ MasterCard	Card Number _____	Exp. Date _____
☐ VISA	☐ American Express	Signature _____	
Name _____		Organization _____	
Mailing Address _____		Phone Number _____	
City _____	State _____	Zip _____	AMOUNT ENCLOSED \$ _____

Mail form to: HCCA4, P.O. Box 428, Los Altos, CA 94023.
Requests for refunds must be made in writing no later than 2/15/89.

Hotel

A block of rooms has been reserved until February 1, 1989 at the Hyatt Regency Monterey (Conference Hotel) and the nearby Monterey Hotel Resort. Attendees should make their own hotel reservations by calling:

Hyatt Regency Monterey In California 1-800-826-2196 Single or Double: \$118
Outside California 1-800-228-9000
Monterey Hotel Resort In California 1-800-622-2099 Single or Double: \$66
Outside California 1-800-426-1685

For additional information or exhibit information please contact: Bill Hickey, Conference Administrator, (415) 969-6920.



HCCA4 Host Institution

Sandia National Laboratories

Sponsors

U.S. Department of Energy, Applied
Mathematical Sciences Program

Strategic Defense Initiative
Organization, Office of Innova-
tive Science and Technology

Joint Tactical Fusion Program Office

U.S. Air Force, Electronic Systems
Division

Air Force Office of Scientific Research
NASA

Issues in Ada Compiler Technology

Mahadevan Ganapathi and Geoffrey O. Mendal
Stanford University

Ada is becoming a language of choice for large software projects, but compilers and other language tools may not have kept pace. This article discusses the key technical issues involved in producing high-quality Ada compilers and related support tools. It also addresses some important problems that compiler designers face — for example, determining which deficiencies of existing Ada systems can be attributed to the language and which are simply hard-to-implement features or unresolved issues in Ada compiler technology.

Technical issues in compiling Ada

Designing and implementing an Ada compiler is a formidable task. But it provides some unique opportunities for skillful software engineering, especially in semantics processing and code optimization, because the software engineering methods largely determine the compiler technologies. Opportunities also exist for developing standard application packages. For example, the Numerics Working Group and Ada Run-Time Environments Working Group, both under the ACM SIGAda umbrella, are now producing preliminary standards for numerical algorithms and runtime interfaces.

Compiler technology is inadequate to support many Ada features, but — an encouraging note — the speed of Ada compilers has doubled from the first to the second generation.

General compilation issues. Ada is a module-based language in which the modules are called packages. A typical compilation unit consists of a single package with a visible interface and a hidden implementation.

Compiling Ada code presents major challenges in the semantics phase that increase compiler design complexity and require a sophisticated Ada programming support environment. In Ada, the properties of a package's hidden implementation cannot be "used" by a client of the pack-

age; a client can only rely on the information in the visible interface. This characteristic emphasizes two important concepts: strong typing and visibility. The checking of these two areas is under the auspices of the semantics phase. Although implementing these concepts complicates semantic checking, it simplifies code generation, since semantic information about the program is gathered before code generation begins.

Current software engineering efforts in Ada compiler technology are focusing on uniformity, machine-independence, small size, optimized target code, and a full-fledged debugging environment. Future environments will have to tackle the bigger issues of interchangeable runtime systems, design tools, and proof/verification tools.

Expertise is needed in handling complex scope and visibility rules (see section on semantic issues), code sharing in generic units, and tasking. In the Ada community, there's little doubt about the needs, but considerable debate about who, if anyone, should specify the solutions. Ada critics claim the language does not say enough about these issues, which were purposely left open to interpretation by compiler implementors. Because each implementation has been designed somewhat differently and therefore has vastly different runtime behavior, these critics doubt that

Ada's portability goals can be achieved. Ada proponents, on the other hand, argue that requiring a specific implementation strategy for each issue would lock the language into outdated compiler, operating system, and processor technologies, thereby defeating the language's long-term objectives and assuring its eventual demise. By not specifying the semantics of these complex issues, they say, the language can take advantage of the latest compiler technologies without losing sight of its overall goals.

In the case of overload resolution, it takes extra effort to interface, in concert, the compiler's front end and the runtime support routines. It is the responsibility of the semantics phase to disambiguate all overloaded names. Once disambiguated, the code generator need only query the symbol table for the entity being referenced. This information can be stored within the intermediate representation or within a symbol table. The important point is that the information must be saved so that back-end and runtime tools do not also have to disambiguate uses of overloaded names. Usually, this is not a challenge in Ada compiler design since all overloaded names are resolved by the semantics phase. Subsequent phases are presented with a unique mapping to each distinct name to avoid ambiguities in code generation and linking. However, runtime tools such as a source-level debugger will need to map back from the generated code to the overloaded name.

Generic units allow efficient sharing of code segments, thus hiding many subtleties. For example, common-body implementation is often not time-efficient. Because Ada does not enforce code sharing among instantiations of a generic unit, most implementors have found it easiest to duplicate the code for each instantiation. Some implementors have discovered new technologies for conquering this problem and do implement a large amount of code sharing. One big obstacle in sharing code among instances of a generic unit is assuring proper performance of constraint checks. One instance might allow removing all constraint checks, while another might require that all be performed (see section on constraint checks).

Support for separate compilation is a basic function of an Ada program library. Implementations are required to check the conformance between separately compiled units. To perform these checks, an Ada compiler's front end must save the relevant conformance information in some man-

ner. The abstract notion of a program library provides a place for storing this information, and an implementation's tools usually provide for user manipulation and querying of the program library.

Ada programming support environments (APSEs) are starting to take advantage of incremental compilation strategies. Robust intermediate languages and a well-designed semantics phase can eliminate recompilations for a "local" or upward-compatible change. The most common form of incremental compilation, one which practically every APSE supports, is recompilation of a package body. Because package clients cannot use information local to the package's hidden part, recompilation of these clients is unnecessary when only the hidden part of a package is changed — the interface to the package is still guaranteed to be unchanged. Applying this concept to smaller pieces of an Ada program is not much different. All it requires is interaction between the program library manager, symbol table, and semantics phase.

On multiprocessor systems, APSEs are expected to compile Ada programs in parallel. Parallel compilation can significantly reduce the edit-compile-link cycle associated with changes requiring recompilation of dependent units. Units that are not dependent on each other can be compiled in parallel.

Implementation of Ada exceptions is the biggest obstacle to effective program optimization (see sections on constraint checks and reordering operations). The effect of Section 11.6 of the US Department of Defense *Reference Manual for the Ada Programming Language*,¹ which was supposed to allow for advanced optimizations by compiler back-end tools, has been reduced to only classical optimizations under the simplest circumstances.

Real-time issues. The runtime code that supports tasking may have a greater effect on target code efficiency than the code generator's instruction-selection phase. Indeed, most APSEs draw a clear distinction between the runtime system (RTS) and the code generator. To support multiple runtime requirements, most code generators simply interface to a runtime supervisor. The interface for all variants of the RTS will be the same. Thus, the compiler implementor can provide for selection among various RTS implementations. Allowing the user to make this selection is, obviously, the next step. Allowing the user to integrate his own RTS

with the compiler implementor's delivered product would be the subsequent step.²

The characteristics and requirements of tasking pose new requirements on the target architecture for compilers and the execution system. The Ada tasking model has been criticized as being too awkward to use.³ Tasks are not abstracted in the same manner as packages (only entries, representation clauses, and pragmas may appear in a task specification). Also, the current runtime tools that support debugging and analysis of concurrent Ada programs are minimal at best, but research into specification language-based debugging tools for Ada tasking is progressing.⁴

The semantics of Ada tasking facilities are not rigorously specified.^{1,2} Furthermore, the availability of synchronization primitives is definitely inadequate. There is no critical region mechanism predefined in Ada, and it is debatable whether any user-defined mechanism can be achieved. Thus, efficient implementation of Ada tasking presents a major challenge. Rendezvous and interrupt handling require efficient task scheduling. Although many common cases of tasking interactions can be handled efficiently with in-line code (assuming this issue is not addressed by a separate RTS), such interactions must be optimized in terms of context switches — that is, one task signaling an event to another requiring a lower level signaling mechanism. Task termination rules present another critical, performance-related design issue, since Ada requires simultaneous, distributed termination of tasks when the terminate alternative is used.

Efficiency in these areas is predominantly a software engineering issue. The facilities to monitor tasking can be handled by runtime support routines, but certain features, such as task abortion, may substantially increase RTS complexity and probably overall execution time.

An Ada RTS for embedded systems must be designed with tight real-time constraints — for example, with a limited amount of memory and the RTS in ROM — yet there is no standard Ada kernel architecture. A general model of runtime behavior calls for machine-independence and interface standardization.² However, standardization is frequently at odds with efficiency and choice of operating systems and processors, thereby charging every use with the overhead of the most general case. Perhaps a separate RTS standard, getting all implementors to agree and abide by rules of runtime behavior, will be approved.

A standard, multitasking kernel of real-time executives with mutual exclusion, signaling, and scheduling primitives can be shared among most compilers. Thus, different compilers can generate code for the same processor. Implementation variability would be expressed as compiler-dependent runtime support. Many implementors already provide this capability.

Programming tools and environments.

There has been greater emphasis on developing better tools and environments than on developing new and better programming languages. Ada compiler vendors have found that providing users with a compiler and some hard-copy documentation is not enough. Ada vendors have had to develop more complete environments where the Ada compiler is simply one of many integrated tools.

The most common tools found in current APSEs are the Ada compiler, source-level debugger, program library manager, configuration management tools, and syntax- and semantics-driven editors. Often, one or more intermediate representations (IRs) are used to integrate all the tools. DIANA (Descriptive Intermediate Attributed Notation for Ada)⁵ is an example of an IR that facilitates development of an integrated tool set.⁶

An Ada source-level debugger needs special support for tasking, especially in the interaction with tasks. For example, a user might want to ask why a task has terminated unexpectedly. TSL (Task Sequencing Language) is a language for specifying the intended behavior of an Ada tasking system.⁴ A number of prototype tools, including a source-level debugger, are being designed around TSL and Ada.

Anna (Annotated Ada) is a language for formally specifying the behavior of sequential Ada programs.⁷ Anna is based on first-order predicate calculus and extends the concept of a package to include package states and package axioms. As with TSL, a number of Anna prototype tools are being designed, and an experimental environment based on Anna is being implemented.

Semantic issues. Many difficult special cases in Ada semantics might be avoided by deriving the compiler implementation from a formal definition. Unfortunately, formal methods are not ready for state-of-the-practice use. Efforts to provide a more formal semantics for Ada have conquered only the sequential aspects of the lan-

guage.⁷ But formal methods for Ada tasking are now being investigated.²

One interesting research approach uses formal semantics to construct a semantic interpretation of a program. Denotational semantics mixes both static and execution semantics and represents them in lambda calculus. Using that semantic framework, a compiler can generate a function representing the effect of the input program. In effect, pointers to denotational functions can be viewed as attributes to tree nodes in attribute grammars. That is, the complexity of using the denotational semantics technique approaches that of complex attribute grammars. Each time a lambda calculus reduction is performed, an IR statement is emitted. Many semantic-based approaches have been used and work continues on this topic. At present, these systems can produce prototype implementations of languages, but the resulting implementations are too inefficient for practical use.

The design of the symbol table needs special care in an Ada compiler. Consider the following illegal Ada declaration:

```
X : INTEGER := X; -- Illegal!
```

Use of an identifier within its own declaration is not allowed. Initially, it might appear too trivial to check for such an occurrence. However, consider the following illegal fragment:

```
declare
  X, Y, Z : INTEGER := 2;
begin
  declare
    X, Y : INTEGER := X + Y; -- Illegal!
  begin
    . . .
  end;
end;
```

This case attempts to reference the identifiers *X* and *Y* within their initialization expressions. The problem is that there are identifiers named *X* and *Y* of the same type in an outer scope. This situation presents a special-case, chicken-and-egg problem. If an implementation adds the identifiers *X* and *Y* to the symbol table and then evaluates the initialization expression, a special-case check would be required to ensure that identifiers within the expression are not being circularly elaborated. If an implementation first checks the initialization expression and then adds the identifiers *X* and *Y* to the symbol table, the routine checking the initialization expression might incorrectly find the outer *X* and

Y identifiers. This situation could easily occur, assuming the expression checker simply calls the symbol table and asks for definitions of any directly visible identifiers *X* and *Y*.

Several special cases like this example require making simple algorithms more complex to satisfy a single language rule. Therefore, it is not surprising that today, even with a well-maintained and vast Ada Compiler Validation Capability (ACVC), compiler bugs are being detected at a very high rate.

Type checking is one of the most important aspects of the semantics phase. Ada allows overloading of functions, procedures, task entries, and enumeration literals. Overloading sometimes introduces ambiguities since alternate definitions, often type-compatible, coexist for the same construct. Elaborate overload resolution is needed to disambiguate and determine a unique meaning. Baker⁸ has shown that an unambiguous resolution, if one exists, can be found in a single bottom-up pass over an expression tree.

Type ambiguities usually result from seemingly innocent declarations. Consider the Ada declarations shown in Figure 1. The statement "*YYY := Y & Y & Y;*" is not ambiguous because there exists only one "&" operator that takes two operands of the type *B* yielding a value of the type *B*. Similarly, the statement "*XXX := X & X & X;*" is unambiguous. However, the statement "*YYY := X & X & X;*" is ambiguous for reasons explained below. For every one-dimensional, nonlimited array type, four "&" operators are implicitly defined after the type declaration (see Figure 2). The problem with the last assignment to *YYY* in is that there exists more than one set of "&" operators that yield a value of the type *B* when provided with operands of the type *A*. The expression "*X & X & X*" is therefore ambiguous. In this case, the runtime semantics of the statement do not depend on which set of "&" operators an implementation chooses. But, in general, this is not the case; consequently, such expressions are illegal.

Code optimization. Code optimization has been a popular area of compiler technology research for over two decades. In the 1960s, there were some source-to-source translators, such as IBM's Fortran to PL/I translator, that performed some optimizations. Due to semantic differences between the source and target languages and also due to lack of available

theoretical groundwork at that time, these optimizers were grotesque. The 1970s saw the foundation of theoretical groundwork in code optimization. Practical application of these ideas began primarily in the 1980s. A number of algorithms and optimization techniques are yet to be completely implemented and evaluated. Unfortunately, there are big gaps in the formal foundation of code optimization theory. Although several isolated algorithms exist, these algorithms do not cover system integration. Engineering an optimizing compiler continues to be a difficult and cumbersome task. In addition, with the proliferation of target architectures, code optimization must be designed to be largely target dependent.

In the past decade, code optimization has become increasingly important for two major reasons: time-critical applications and exploration of new architectures. In particular, in the Ada world, embedded systems with time- and space-critical applications require optimizing compilers. At the same time, new architecture technologies, such as reduced-instruction-set architectures, are fundamentally dependent on optimizing compilers.

Constraint checks. Strong typing requires constraint checks to assure that an object's value always belongs to a well-defined set of values. Object code constraint checks can be eliminated by the code generator if it can determine that the checks, if performed, will always be satisfied. Currently, most of the Ada implementor's code optimization efforts go toward eliminating such checks. With Ada's strong typing and advanced compiler technologies such as flow analysis, up to 90 percent of the constraint checks in Ada programs can be eliminated. Clearly, this is one case where increased compiler activity results in more efficient use of machine resources at runtime.

As shown in Figure 3, an optimizing Ada compiler can eliminate all constraint checks on the first assignment, since the subtype of *Char* is a superset of the subtype of *Letter* and because *Letter* must have been assigned a legal value during the call to *Text_IO.Get*. Such a compiler is able to detect this during compilation. The second assignment, however, will require some form of runtime constraint check, since the value of *Letter* might be an "A." When the character "A" is input, this second assignment cannot proceed and must raise the exception *Constraint_Error*.

Although Ada does not require a com-

```
type A is array (POSITIVE range < >) of INTEGER;
subtype Sub_A is A (1 .. 1);
type B is array (POSITIVE range < >) of Sub_A;

X    : A (1 .. 1);
XXX  : A (1 .. 3);
Y    : B (1 .. 1);
YYY  : B (1 .. 3);
```

Figure 1. Ada declarations leading to type ambiguities.

```
function "&" (Left : the_array_type;      Right : same_array_type)
return same_array_type;

function "&" (Left : the_array_type;      Right : the_component_type)
return same_array_type;

function "&" (Left : the_component_type;   Right : the_array_type)
return same_array_type;

function "&" (Left : the_component_type;   Right : the_component_type)
return the_array_type;
```

Figure 2. Implicitly defined "&" operator specifications.

```
with Text_IO;
procedure Constraint_Checks is
  subtype Upper_Case is Character range 'A' .. 'Z';
  subtype Bees       is Character range 'B' .. 'b';

  Letter : Upper_Case;
  Bee    : Bees;
  Char   : Character;
begin
  Text_IO.Put ("Enter a letter: ");
  Text_IO.Get (Letter);
  . . .

  Char := Letter; -- 1.
  Bees := Letter; -- 2.
end Constraint_Checks;
```

Figure 3. Constraint check elimination.

```

with Text_IO;
procedure Reorder is
  X, Y : Integer := 1;
begin
  X := 2;           -- 1.
  Y := Integer'Last + 1; -- 2.
exception
  when Numeric_Error | Constraint_Error =>
    Text_IO.Put_Line (Integer'Image (X) & Integer'Image (Y));
end Reorder;

```

Figure 4. Semantics of reordering operations.

piler to check for every conceivable programming error (such as reference to an uninitialized variable), it does not prevent advanced tools from discovering such errors. The language classifies a program's execution as *erroneous* when all required language rules have been satisfied but the program's runtime behavior still violates some rules not enforced by the Ada Standard. For example, a program that reads the value of an object before a value has been assigned to the object is said to execute erroneously. Advanced tools and environments are currently being designed to detect and more predictably report on the runtime behavior of erroneous programs.

Embedded checks pose problems for code motion and reduction in operator strength. Constraint checks impose performance degradation in the target code. In the worst case, checks after every arithmetic operation and checks for interrupt overflow after every instruction would render the target code unusable. Optimization of constraint checks is therefore essential. For example, hardware interrupts should be fast. In some cases, target architectures provide an interrupt-on-overflow instruction to accomplish this optimization efficiently.

Reordering operations. Reordering of operations (code motion) is important for effective code optimization, especially for pipeline and multiprocessor target systems. The intent of Section 11.6 of the reference manual¹ was to give implementors the flexibility needed for such optimizations. But, after some experience with

the language, users and implementors are discovering that Section 11.6 is more of a hindrance than a help in solving these optimization problems. A myriad of special cases effectively negate all attempts to reorder operations in an Ada program. The Ada Rapporteur Group is collecting all pertinent ideas, but at the time of this writing, no resolution has been reached.

To understand why the reordering of operations is such a delicate issue, consider the Ada program shown in Figure 4. The program can print either 2 1 or 1 1 because an implementation is free to reorder the evaluation of the right-hand expressions of assignment statements. If the expression in statement number two is evaluated first, and this evaluation raises *Constraint_Error* or *Numeric_Error*, the program will print 1 1. If the program is executed in the canonical order, or if the evaluation of the expression in statement number two does not raise an exception, then 2 1 will print. Note that *Constraint_Error* may be raised after the expression in statement number two has been successfully evaluated, but before the assignment to *Y* takes place. In this case, it is not the evaluation of the expression that raises the exception, but rather the constraint check required by the assignment operation.

Design of optimizing Ada compilers. In a production-quality compiler, optimizations are usually performed continuously at various levels — for example, source-to-source transformations, intermediate-representation optimizations, and post-pass optimizations.

Source-to-source transformations include

- in-line expansion,
- eliminating tail recursion, and
- eliminating recursion with the help of Ada tasking.

A production-quality compiler could have two intermediate representations, one for machine-independent optimizations and another for retargetable code generation (e.g., DIANA,⁵ Ucode, and 1R⁹). Intermediate representation optimizations include

- common subexpression elimination,
- code motion and reordering optimizations,
- dead-code elimination,
- strength reduction,
- induction variable elimination,
- constant and value propagation (copy propagation),
- assertion propagation,
- global register allocation, and
- interprocedural optimization.

Post-pass optimizations include

- machine-dependent optimization,
- peephole optimization,
- reordering for pipeline constraints, instruction scheduling and branch delays for increased performance, and
- scheduling for microcode compaction.

To explore the effectiveness and limitations of code optimization, the proposed optimization techniques must be completely implemented and evaluated. But practical application of code-optimization concepts began in this decade only — and seldom in the case of Ada compilers. Those few instances are inadequate for forming a considered opinion.

Integrating code-optimization algorithms in a compiler requires substantial effort, and certain language issues present additional difficulties. In Ada, unlike other high-level languages, a code optimizer has extra freedom to alter the semantics of legal programs while preserving execution semantics. However, this limited license to enhance performance disallows potentially important optimizations. For example, the optimizer has to account for complex interactions with scoping rules and also limit code motion. Fortunately, most constraints on peephole optimizations can be removed with the help of an integrated code-generator peephole-optimizer package that makes wider contextual information available.¹⁰

To handle numerous special cases, an optimizer must conform to the language's semantic subtleties. At the same time, optimizations performed during machine-code generation should be completely source-language independent. Thus, complex visibility rules are resolved before code reaches the optimization phase, but generic units and in-line code depend on optimization techniques such as constant folding and dead-code elimination. Ada-specific optimizations that are difficult to implement include reordering of operations, exceptions, and task interactions. Optimizing runtime support for tasking, for example, may require producing in-line procedure code to replace entry calls — a possibility, since Ada tasks need not be implemented as processes.

Exception handling. Exception handling is a complex and challenging feature in Ada. An exception can be raised implicitly as well as explicitly; furthermore, its propagation properties can be static to containing blocks and packages or dynamic to a subprogram caller. Propagation rules follow complex scoping rules, and an exception can propagate to a scope where the exception name is not visible. Since exception handling has significant runtime implications, it is important to reduce unnecessary runtime constraint checking and implement active, efficient exception handlers.

In the presence of exception semantics, code generation and optimization are constrained to a certain extent. Such constraints were identified as early as the Fortran H compiler. For example, consider dead-code elimination in the presence of exception propagation and exception handlers. The legality of operation reordering varies depending on whether an exception is raised in the presence of code motion or regardless of it. It is illegal to modify the value of a visible variable before an exception is raised when that variable should have been modified only after the statement that raised the exception. When rearranging assignment statements, exceptions may be reordered subject to the constraint that the same exception handlers are invoked. Other such limitations to code reordering of assignment statements can restrict other optimizations such as vectorization of loops and can seriously degrade performance on pipelined architectures.

Aliasing. Aliasing is a language feature that has always had detrimental effects on

Aliasing is a language feature that has always had detrimental effects on code optimization.

code optimization, but accurately assessing these effects is a complex issue. In an attempt to limit aliasing in Ada, all aliases except access values are declared explicitly.¹¹ Furthermore, compared to less strongly typed languages, such as C and PL/I, Ada's access values simplify explicit alias analysis. Because of the semantics of mode out and in out parameters (by copy-restore or reference), the decision of value or reference passage is left to the compiler. Thus, aliasing between a reference-parameter and a global-variable or between a reference-parameter and another reference-parameter is erroneous. To eliminate this class of erroneous execution, Ada access values are always passed by copy. In principle, one could ignore the possibility of pass by reference when writing a compiler, but this would not be practical for efficiency reasons. In Ada, it is easy to ascertain that mode out and in out parameters are indeed assigned. Thus, in this area, a global optimizer for Ada should be able to perform better than one for C.

Areas for further study. Effective use of interprocedural optimizations and issues such as exception handling, separate compilation, debugging, program development environments, and automatic parallelization remain to be fully investigated. Optimizations across package boundaries and Ada-specific link-time optimizations are just beginning to be investigated.

Ada compiler technology

At least two issues in Ada compiler technology — lexical and syntax analyses — can be considered solved problems. Parser constructors can process grammars to

automatically produce parsers for a wide class of grammars. Even automated error-recovery techniques are feasible. However, compiler design must respond to the requirements of new programming languages and machine architectures.¹² Data abstraction languages offer challenges in the semantics phase, as well as requiring new techniques for code generation and effective optimization strategies.

In the last decade, compiler implementors have begun serious study of retargetable compilers, emphasizing minimal recoding, incremental development, increased reliability, and improved performance. Some important problems remain unsolved in code generation — for example, matching expressions containing common subexpressions. Research continues in automation of back-end functions, including code optimization.

The major thrusts of current research in compiler technology are

- automatic generation, to reduce the effort needed to write a compiler, and
- code optimization, to reduce the execution time and memory requirements of the object program.

Important compilation issues in Ada include

- intermediate program representations,
- code optimization,
- integration with an Ada programming support environment,
- correctness and quality control,
- implementation technologies such as garbage collection, tasking primitives, I/O, and data representation features to take advantage of language semantics, and
- producing runtime systems for different applications — for example, a bare machine implementation.

Some critics have branded Ada “too slow” or “not proper” for current applications. Although the *language design* may be the perceived cause, it is in fact the current *implementations* that are to blame. Making efficient use of machine resources is a challenge for Ada implementations. Meeting that challenge requires better application of known technologies.

One of the most important technologies is intermediate representations, and all Ada language tools use one. IRs meet Ada's strong typing and conformance requirements, especially across separately compiled units, and provide a foundation for building well-integrated Ada programming environments.

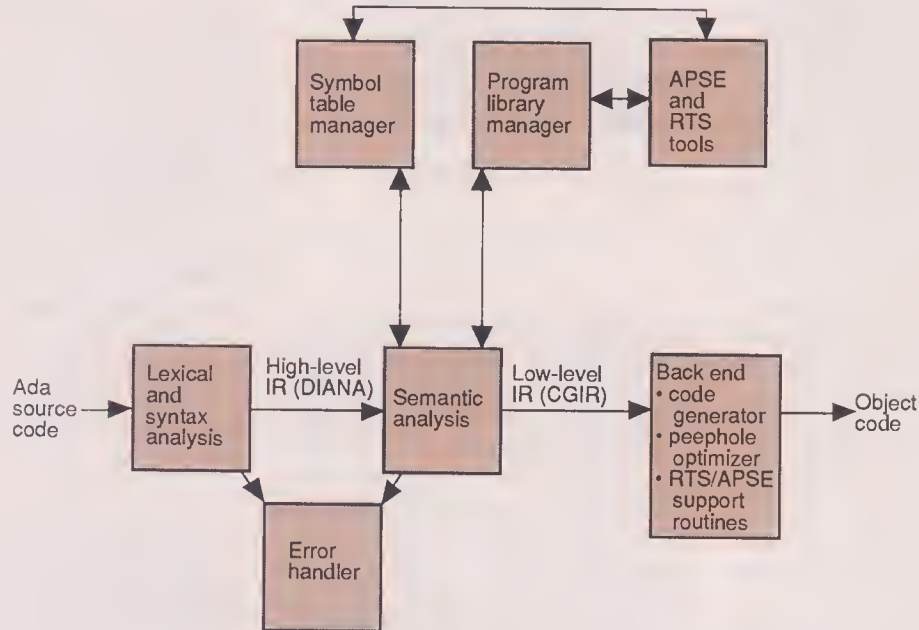


Figure 5. Structure of a typical Ada compiler.

Structure of an Ada compiler. Although the individual pieces of an Ada compiler tend to be more complex than those for other high-level languages, its overall structure follows the classical model of a compiler for any high-level language.

In a typical Ada compiler (see Figure 5), lexical and syntax analyses transform Ada source code to a high-level IR. Many Ada compilers use a tree-based IR such as DIANA. The semantic analysis phase then populates the high-level IR with the necessary information and outputs the low-level IR required by the back end (code generator, peephole optimizer, RTS, APSE tools). As stated earlier, the design of the symbol table manager and program library manager requires some care, since both components must integrate with the semantic analysis and code generator phases. Other APSE tools, such as a source-level debugger and cross-reference utility, must also interface with the symbol table manager and program library manager.

Special problems in compiling Ada. Ada compilers must be designed as envi-

ronment tools, not stand-alone programs, because a fast Ada compiler that generates good code solves only half the problem. Developing large Ada programs requires other tools that would be inappropriate lumped into a stand-alone compiler. For this reason, Ada compilers must be tightly integrated into an APSE and, as compiler vendors have discovered, cannot be marketed successfully without one. Compilers for other high-level languages are typically designed, implemented, and sold as stand-alone products; other products can be purchased ad hoc to form a development environment. Ada does not lend itself well to this philosophy. An Ada compiler typically forms the core of an APSE, driving the design and implementation of other APSE tools. Currently, the state of Ada compiler technology precludes purchasing a compiler from one vendor and an APSE tool such as a source-level debugger from another.

The complexities of an Ada compiler are not caused by the number of language rules that must be satisfied, but rather by the complex interrelationships between these rules. Overload resolution, code

sharing for generic units, tasking, and code optimization are some examples. Overload resolution is considered a solved problem, yet compiler bugs in this area are still dominant. Code sharing for generic units can be handled easily for some generic units, but a general model that will satisfy all generic units has yet to emerge. The semantics of the tasking model leave too much room for implementors to do as they please. Finally, code optimization is hindered by specific language rules that are so broad as to make it extremely difficult to formulate an optimization strategy for a specific target architecture.

Outlook for better implementations. Ada is an ANSI standard, strongly supported by the US Department of Defense, but many implementation problems remain. Below, we offer suggestions that should lead to much better implementations.

Use of the following Ada capabilities may hinder optimizations currently implemented in most Ada compilers:

- exceptions,
- generic units,

- private and limited private types in package definitions,
- naming and modularity of package definitions, and
- tasking.

Issues that continue to concern the Ada community include

- standardization of DIANA,
- objects and object-oriented methodologies,
- inheritance capabilities for (parent) processes, and
- organization of the runtime structure, for example, hierarchical, using threads of control.

Although over 200 validated Ada compilers are available for more than 25 computer architectures, compiler technology has been inadequate to support many of Ada's features. For example, many compilers support error reporting without raising of exceptions in the compiled programs. In general, these compilers have been large and quite slow. They execute approximately 1,000 lines per minute for syntax checking. Much of this weakness can be attributed to implementation strategies for program libraries and symbol tables. Because of hardware dependency, the runtime library cannot be 100-percent portable. Furthermore, ACVC testing has revealed only a small number of implementation dependencies. Machine-dependent Ada features such as machine-code insertions or assembly-language subprograms must be discouraged. They must be isolated using packages or Ada's predefined interfacing mechanism, pragma Interface. It is encouraging to note that second-generation Ada compilers are typically twice as fast as their first-generation counterparts. However, space efficiency can be further improved with compact data representations.

The following list highlights directions needed to improve the current state of Ada compiler technology:

- new technologies (compilers for Ada take longer to mature),
- robust APSEs,
- efficient host/target architecture bindings,
- emphasis on usability, not simply on validation,
- execution speed (200 lines per minute for optimized object code), and
- execution quality (at most, 30 percent space-efficient, 15 percent time-efficient).

After obtaining validation certificates, compiler vendors are now concentrating

Intrinsic parts of the Ada language require substantial processing time that cannot be avoided.

on improving performance, especially in code optimization and specific compilation issues for real-time and distributed computing. Because of semantic issues involving timing, exceptions, abortion, and priorities, rendezvous abstractions become inherently inefficient with potentially substantial implementation overhead. Thus, intrinsic parts of the Ada language require substantial processing time that cannot be avoided. They continue to compromise real-time performance. With the ongoing interchanges in hardware/software trade-offs for improved performance, the future may reveal specialized hardware to alleviate some of these problems.

Consider parallel and distributed target environments. It is evident that tasking and exception handling do not map easily to most current distributed operating systems. To compensate for the inadequacies of the underlying operating system, compiler writers need to produce grafted runtime support. These runtime environment services include dynamic storage management, virtual shared memory services, communication facilities, clocks and delays, and additional services such as incremental library functions. Alternately, these services must be placed in the typical distributed/parallel operating system. Furthermore, to accommodate tasking and extensibility in a large number of implementations, large code segments must be sacrificed to the runtime system along with execution speed, reliability, and security risk. Unlike a general-purpose operating system, the target environment cannot tolerate such overheads. Thus, vendors must support a "bare machine" philosophy with the runtime system supported by a kernel that must encapsulate target-dependent features such as I/O services, memory management, process management, and interrupt handling.

Calls for runtime services and resources depend on the underlying system software to manage all services and resources shared by separate application programs. Thus, systems software based on Ada implementations that manage shared resources is provided and must support multiple, independent application programs that use Ada services.

The Ada user community is currently focusing on ACVC conformance and pure compilation throughput. A big issue of the future is interchangeable runtime systems, either defined or settable by the user. Currently, most APSEs allow the user to interface with the target environment by means of a package that invokes system calls. The runtime system provided by the vendor has no such interface. The user has few alternatives for changing the performance characteristics of the runtime system. What users will want in the near future is the ability to tailor the vendor's runtime system or even substitute one of their own.

Ada has not yet been accepted for embedded, real-time systems because of significant implementation problems. Fundamentally, it does not provide careful and precise definitions of program distribution semantics, reconfiguration, and recovery. Multiple choices exist for specification and implementation of dynamic reconfiguration, partitioning, configuration management, and granularity of fault-tolerance. Distributed execution of real-time programs, implementation of tasking and interrupt handlers, and related optimizations continue to pose a real challenge. In a distributed environment, the fundamental unit of distribution can range from variables and/or tasks to packages. Furthermore, many such units may share resources and communicate with other units by any one of the following object-level mechanisms: nested transactions, virtual shared memory, asynchronous send-receive protocols, remote procedure calls, and task rendezvous across the network. Distribution can be specified using pragma constructs, library subprograms, or library packages.

Integrating a compiler in a new environment remains a challenging task. Rehostability (hosting software on a different operating system) is a more critical issue than retargetability (generating code for a different machine architecture). Current implementations have largely ignored embedded applications that impose special requirements on implementations. For example, fault-tolerant computing requires careful design decisions that are

realized as embedded systems programming specifications:

- responses to hardware interrupts; interrupt handling efficiency, special pragmas; many compilers do not support interrupt handling;
- memory-mapped I/O; special locations;
- ROM-able code;
- embedded or mission-critical instruction-set-architecture targets; and
- reconfigurable runtime support; tuning algorithms for scheduling and storage allocation.

It is difficult to assess the state of a technology, even a very specific one — namely, compilers for Ada. We purposely refrained from making vendor-specific comparisons. Although we may have implied that Ada compilers are “birds of a feather,” it is actually the compiler technologies — not the various products — that are the same. As usual in software engineering, overall product quality depends more on the structure of the compiler and the internal data than on the particular problem-solving method. Additionally, for an Ada compiler, the way it’s embedded in the environment is crucial to its performance and ease of use.

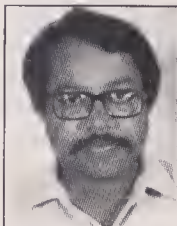
Achieving a validation certificate (a time-consuming effort) does not imply product quality for practical applications — that requires a much more commercial quality-assurance scheme than the ACVC. The ACVC tests only for conformance to the language standard, not for meeting real constraints on resource use, and users are beginning to understand that validation does not guarantee the compiler will meet project requirements. So far, development of Ada compilers has been acceptable, but implementors who wish to remain competitive in Ada markets will soon be forced to provide APSEs that are broader in scope, more efficient, and of higher quality. □

Acknowledgments

The authors wish to thank David Rosenblum and Doug Bryan for their persistence in reviewing many drafts of this article. Also, the anonymous *Computer* referees made many insightful comments that led us to clarify and rethink many of our arguments.

References

1. *Reference Manual for the Ada Programming Language*, US Dept. of Defense, US Govt. Printing Office, ANSI/MIL-STD-1815A edition, Jan. 1983.
2. D.S. Rosenblum, *Design and Verification of Distributed Tasking Supervisors for Concurrent Programming Languages*, PhD Thesis CSL-TR-88-357, Stanford Univ., Mar. 1988.
3. *SIGAda Int'l Workshop on Real-Time Ada Issues*, ACM, New York, 1987.
4. D.C. Luckham et al., *Task Sequencing Language for Specifying Distributed Ada Systems, TSL-1*, Tech. Report CSL-TR-87-334, Stanford Univ., July 1987.
5. G. Goos et al., *An Intermediate Language for Ada*, Vol. 161 of *Lecture Notes in Computer Science*, Springer-Verlag, 1983.
6. D.S. Rosenblum, “A Methodology for the Design of Ada Transformation Tools in a DIANA Environment,” *IEEE Software*, Vol. 2, No. 2, Mar. 1985, pp. 24-33.
7. D.C. Luckham et al., *Anna, a Language for Annotating Ada Programs*, Vol. 260 of *Lecture Notes in Computer Science*, Springer-Verlag, 1987.
8. T.P. Baker, “A One-Pass Algorithm for Overload Resolution in Ada,” *TOPLAS*, Vol. 4, No. 4, Oct. 1982, pp. 601-614.
9. M. Ganapathi and C.N. Fischer, “Attributed Linear Intermediate Representations for Retargetable Code Generators,” *Software — Practice and Experience*, Vol. 14, No. 4, Apr. 1984, pp. 347-364.
10. M. Ganapathi and C.N. Fischer, “Integrating Code Generation and Peephole Optimization,” *Acta Informatica*, Vol. 25, No. 1, Jan. 1988, pp. 85-109.
11. G.O. Mendal, “Storage Reclamation for Ada Programs,” *Using Ada: ACM SIGAda Int'l Conf.*, ACM, New York, pp. 180-189.
12. J.L. Hennessy and M. Ganapathi, “Advances in Compiler Technology,” *Ann. Rev. of Computer Science*, 1986.



Mahadevan Ganapathi is affiliated with the Computer Systems Laboratory at Stanford University. His interests are in programming languages, compiler construction, and computer architecture. His specific efforts have been on the design of intermediate representations, code generation, and code optimization with additional interests in the theory of parsing and translation, attribute grammars, and attributed parsing. Related research involves retargetable code generation and peephole optimization algorithms. His recent efforts include interprocedural analysis and optimization, and also compilers for functional languages.

Ganapathi obtained his bachelor's degree in electronics and communication engineering from the Indian Institute of Technology in 1976 and his PhD in computer sciences from the University of Wisconsin, Madison, in 1980.



Geoffrey O. Mendal is affiliated with the Computer Systems Laboratory at Stanford University. His interests are in programming languages and language tools. Currently, he is developing compiler tools for the Annotated Ada (Anna) specification language project. He is currently the technical editor of the ACM SIGAda publication *Ada Letters*.

Mendal is a member of ACM and the IEEE Computer Society. He received an IEEE Computer Society Outstanding Contribution Award in April 1986 for Best Paper at the Second International IEEE Computer Society Conference on Ada Applications and Environments. He received his bachelor's degree in computer and communication sciences from the University of Michigan in December 1983.

The authors' address is Computer Systems Lab, Stanford University, Stanford, CA 94305.

Software through PicturesTM

"Delivering the Promise of..."

HIGH QUALITY SOFTWARE is the promise of CASE.

IDE's Software through Pictures increases productivity, establishes control, and improves communications. Everyone benefits when software is delivered on-time and within budget and when it matches the customer's needs. In addition, projects designed right the first time have significantly lower maintenance costs.

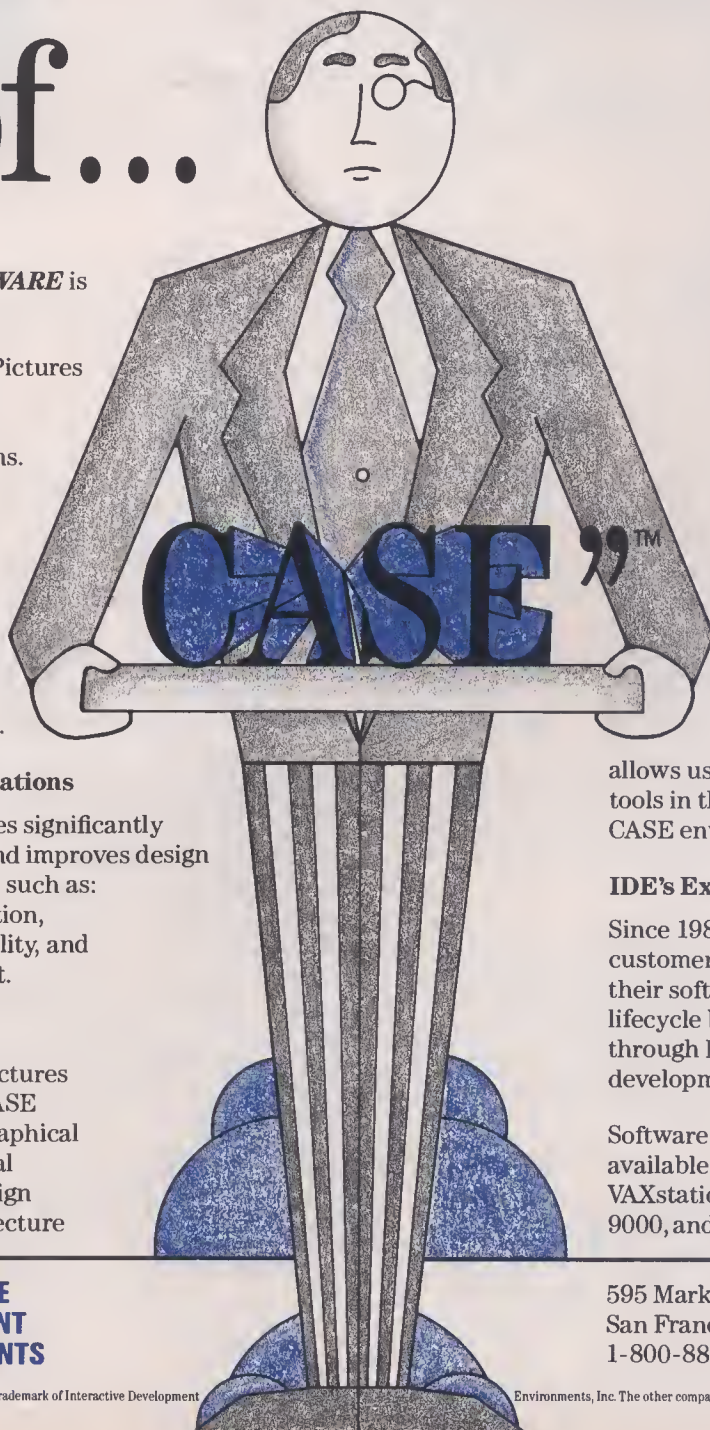
User-Extensible Applications

Software through Pictures significantly increases productivity and improves design quality with applications such as:

- Automatic Documentation,
- Requirements Traceability, and
- DOD-STD-2167 Support.

Open Architecture

The Software through Pictures integrated, multi-user CASE environment includes graphical editors supporting several popular analysis and design methods. Its open architecture



Free IDE Seminars

Call 1-800-888-IDE1 ext. 121 today to register.

AL	Huntsville	Feb. 15
AZ	Phoenix	Mar. 3
CA	Anaheim	Mar. 22
	El Segundo	Jan. 25
	Ridgecrest	Feb. 1
	San Diego	Feb. 3
	San Fernando Valley	Feb. 15
	San Francisco	Mar. 28
	Sunnyvale	Feb. 21
CO	Denver	Jan. 31
CT	Stamford	Jan. 23
DC	Washington	Jan. 20,
		Feb. 17, Mar. 17
FL	Ft. Lauderdale	Jan. 27
	Orlando	Jan. 13
	Tampa	Mar. 22
GA	Atlanta	Mar. 2
IL	Chicago	Mar. 15
MA	Boston	Feb. 7
MD	Baltimore	Jan. 10,
		Feb. 16, Mar. 16
MI	Detroit	Jan. 26
MN	Minneapolis	Feb. 23
NC	Raleigh	Jan. 25, Feb. 24
NJ	Parsippany	Feb. 6
	Princeton	Mar. 16
NM	Albuquerque	Mar. 31
NY	Long Island	Feb. 22
	New York City	Jan. 9
	Rochester	Feb. 1
OH	Cincinnati	Mar. 23
	Cleveland	Feb. 8
	Columbus	Jan. 24
PA	Philadelphia	Jan. 10
	Pittsburgh	Jan. 31
TX	Austin	Mar. 2
	Dallas	Feb. 9
	Ft. Worth	Feb. 16
	Houston	Jan. 26, Mar. 17
UT	Salt Lake City	Feb. 24
WA	Seattle	Feb. 2
ONT	Toronto	Mar. 21

allows users to incorporate other tools in their own integrated CASE environment.

IDE's Experience

Since 1983, IDE has been helping customers succeed at each stage of their software development lifecycle by incorporating Software through Pictures into their development environments.

Software through Pictures is available for the Apollo, DEC VAXstation (VMS and Ultrix), HP 9000, and Sun Workstations.

**INTERACTIVE
DEVELOPMENT
ENVIRONMENTS**

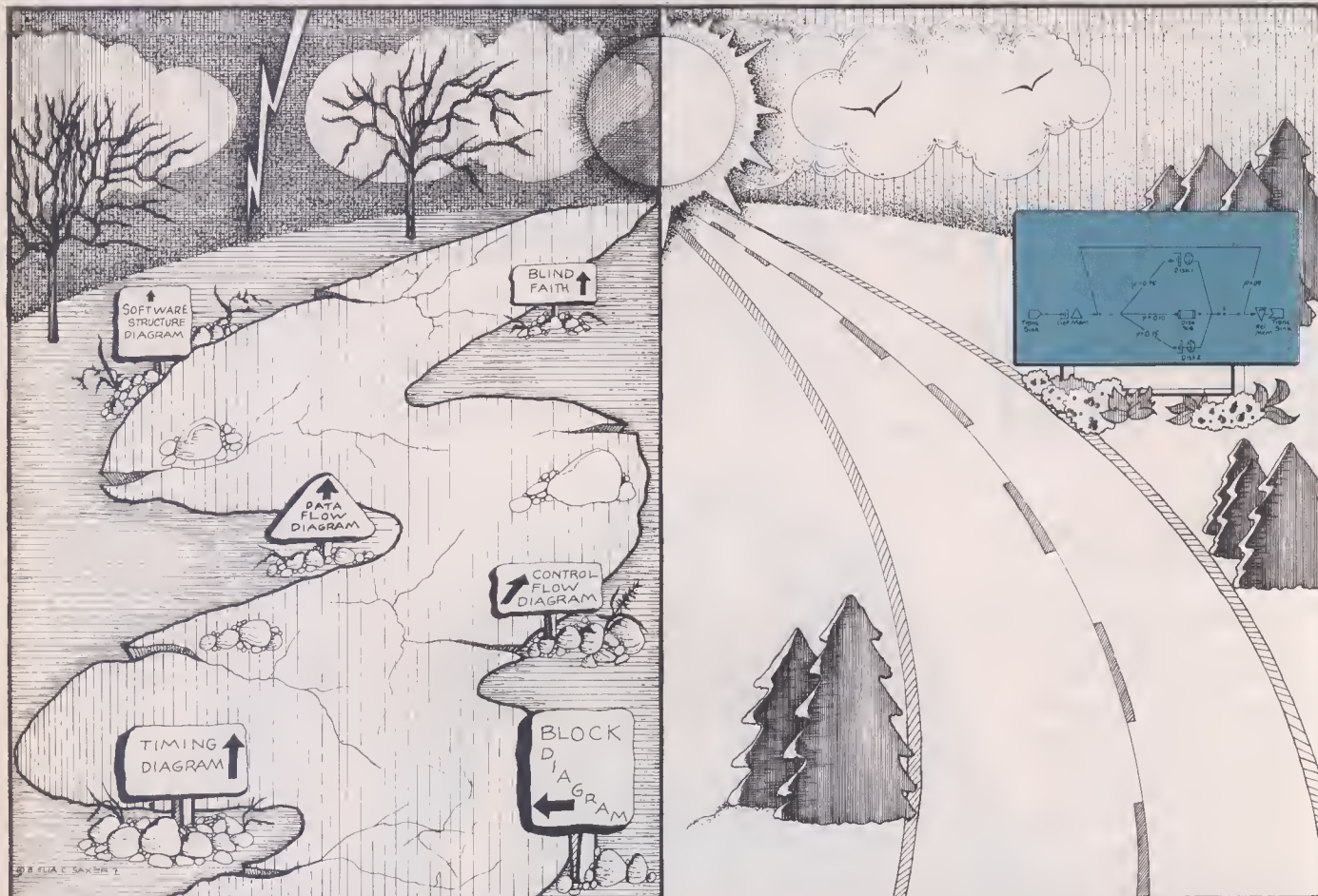
© 1988 IDE. Software through Pictures is a trademark of Interactive Development

595 Market Street, 12th Floor
San Francisco, California 94105
1-800-888-IDE1 or 1-415-543-0900
OUTSIDE U.S.

Environments, Inc. The other companies mentioned own numerous registered trademarks.



GET ON THE "PAWS™ EXPRESSWAY"



TO SYSTEM DESIGN SOLUTIONS

Why not take the shortest and fastest route toward system design solutions? To define and model a large, complex system, you need a modeling tool that can address the system's total architecture.



PAWS/GPSM™ is an easy-to-use high productivity modeling and simulation tool that lets you evaluate alternative architectures while focusing on performance. It provides a state-of-the-art environment for developing accurate and reliable product definitions.



PAWS provides high level primitives closely resembling the user's intuition. PAWS models the behavior of a total system implemented in diverse technologies including software, hardware, and firmware.



GPSM, the graphical interface to PAWS, helps you quickly and easily transfer your ideas into pictures. GPSM helps you visualize multiple data and control flows through several components and incrementally synthesize models of a total system.



PAWS/GPSM is flexible. It lets you refine your model as your system design evolves so you can eliminate potential problems as early as possible in the product design cycle.

Call (512) 474-4526 to receive information on PAWS/GPSM. And get on the PAWS expressway to system design solutions.

PAWS and GPSM are trademarks of Information Research Associates, Inc.



Scientific and Engineering Software, Inc.
1301 West 25th Street Suite 300 Austin, Texas U.S.A. 78705 512/474-4526
Formerly Information Research Associates

Reader Service Number 7

Computing as a Discipline

Final Report of the ACM Task Force on the Core of Computer Science, in cooperation with the IEEE Computer Society

Peter J. Denning (chair), Douglas E. Comer,
David Gries, Michael C. Mulder, Allen Tucker,
A. Joe Turner, and Paul R. Young

As ACM enters its 42nd year, an old debate continues. Is computer science a science? An engineering discipline? Or merely a technology, an inventor and purveyor of computing commodities? What is the intellectual substance of the discipline? Is it lasting or will it fade within a generation? Do core curricula in computer science and engineering accurately reflect the field? How can theory and lab work be integrated in a computing curriculum? Do core curricula foster competence in computing?

We in the field project an image of a technology-oriented discipline rooted in mathematics and engineering. For example, we represent algorithms as the most basic objects of concern, and programming and hardware design as the primary activities. The view that "computer science equals programming" is especially strong in most of our curricula: programming constitutes the introductory course, technology appears in the core courses, and science goes into the electives. This view blocks progress in reorganizing the curriculum and diverts the best students, who want a greater challenge. It denies a coherent approach to making experimental and theoretical computer science integral and harmonious parts of a curriculum.

Those in the discipline know that computer science encompasses far more than programming. For example, designing hardware, formulating system architectures, designing operating system layers, structuring a database for a specific application, and validating models belong to the discipline, but are not programming. The emphasis on programming arises from our long-standing belief that programming languages are excellent vehicles for gaining access to the rest of the field—a belief that limits our ability to speak about the discipline in terms that reveal its full breadth and richness.

The field has matured enough that we can now describe its intellectual substance in a new and compelling way. This realization arose in discussions among the heads of the PhD-granting departments of computer science and engineering at their meeting in Snowbird, Utah, in July 1984. The discussions inspired interest in the ACM and the IEEE Computer Society to form task forces chartered to create a new approach. In the spring of 1985, ACM President Adele Goldberg and ACM Education Board Chair Robert Aiken appointed this task force on the core of computer science with the enthusiastic cooperation of the IEEE Computer Society. At the same

time, the Computer Society formed a task force on computing laboratories with the enthusiastic cooperation of the ACM.

We hope that the work of the ACM task force, embodied in this report, will produce benefits beyond the original charter. By identifying a common core of subject matter, it can catalyze the coalescing of the processes for developing curricula and model programs in the two societies. It can serve as the basis for future discussions of computer science and engineering as a profession. It can stimulate improvements in secondary school courses in computing. And, it can lead to a greater appreciation of computing as a discipline by those outside the field.

We set for ourselves the goal of creating a new way of thinking about the field. Hoping to inspire general inquiry into the nature of our discipline, we sought a new intellectual framework, not a prescription. We invite you to adopt the framework and adapt it for your own curricula.

Charter of the task force

The task force had three general charges:

(1) Describe computer science in a way that emphasizes fundamental questions and

significant accomplishments. The definition should recognize the constantly changing nature of the field and point out that what is said is merely a snapshot of an ongoing process of growth.

(2) Propose a teaching paradigm for computer science that conforms to traditional scientific standards, emphasizes the development of competence in the field, and harmoniously integrates theory, experimentation, and design.

(3) Give a detailed example of an introductory course sequence in computer science based on the curriculum model and the disciplinary description.

We immediately extended our task to encompass both computer science and computer engineering, for we concluded that the core material for the two fields shows no fundamental difference. The differences manifest in the way the two disciplines elaborate the core according to different standards of competence: computer science focuses on analysis and abstraction, computer engineering on abstraction and design. We use the phrase "discipline of computing" here to embrace all of computer science and engineering.

Two important issues lie outside the charter of this task force. First, the curriculum outline we created deals only with the introductory course sequence. It does not address the important, larger question of the design of the entire core curriculum. Nonetheless, the suggested introductory course would be meaningless without a new design for the rest of the core. Second, our specification of an introductory course is intended as an example approach to introduce students to the whole discipline in a rigorous and challenging way, an "existence proof" that our definition of computing can be put to work. We leave it to individual departments to apply the framework to develop their own introductory courses.

Paradigms for the discipline

As a context for our definition of the discipline of computing, it helps to make explicit the three major paradigms, or cultural styles, by which we approach our work.

The first paradigm, theory, is rooted in mathematics and consists of four steps followed in the development of a coherent, valid theory:

- characterize objects of study (definition),
- hypothesize possible relationships among them (theorem),
- determine whether the relationships are true (proof), and
- interpret results.

A mathematician expects to iterate these steps, for example, when errors or inconsistencies are discovered.

The second paradigm, abstraction (modeling), is rooted in the experimental scientific method. It consists of four stages followed in the investigation of a phenomenon:

- form a hypothesis,
- construct a model and make a prediction,
- design an experiment and collect data, and
- analyze results.

A scientist expects to iterate these steps, for example, when a model's predictions disagree with experimental evidence. Even though "modeling" and "experimentation" might be appropriate substitutes, we have chosen the word "abstraction" for this paradigm because this usage is common in the discipline.

The third paradigm, design, is rooted in engineering. It consists of four steps followed in the construction of a system (or device) to solve a given problem:

- state requirements,
- state specifications,
- design and implement the system, and
- test the system.

An engineer expects to iterate these steps, for example, when tests reveal that the latest version of the system does not satisfactorily meet the requirements.

Theory is the bedrock of the mathematical sciences: applied mathematicians share the notion that science advances only on a foundation of sound mathematics. Abstraction (modeling) is the bedrock of the natural sciences: scientists share the notion that scientific progress is achieved primarily by formulating hypotheses and systematically following the modeling process to verify and validate them. Design is the bedrock of engineering: engineers share the notion that progress is achieved primarily by posing problems and systematically following the design process to construct systems that solve them.

Many debates about the relative merits of mathematics, science, and engineering implicitly rely on an assumption that one of the three processes (theory, abstraction, or design) is the most fundamental. However, a closer examination reveals that, in computing, the three processes so intricately intertwine that it is irrational to say that any one is the most fundamental. Instances of theory appear at every stage of abstraction and design, instances of modeling at every stage of theory and design, and instances of design at every stage of theory and abstraction.

Despite their inseparability, the three paradigms represent distinct areas of competence. Theory concerns the ability to describe and prove relationships among objects. Abstraction concerns the ability to use those relationships to make predictions that can be compared with the world. Design concerns the ability to implement specific instances of those relationships and use them to perform useful actions. Applied mathematicians, computational scientists, and design engineers generally do not have interchangeable skills.

What is more, in computing we tend to study computational aids that support people engaged in information-transforming processes. On the design side, for example, sophisticated VLSI design and simulation systems enable the efficient and correct design of microcircuitry, while programming environments enable the efficient design of software. On the modeling side, supercomputers evaluate mathematical models and make predictions about the world, while networks help disseminate

Complete ACM task force report available

An unabridged edition of the body and first appendix of this report was published in the January 1989 issue of *Communications of the ACM*.

The complete report includes two appendixes. The first consists of a full definition of computing as a discipline. A condensed version of Appendix I, based on the matrix framework proposed by the task force, appears on pages 68 and 69 of this issue of *Computer*. Appendix II of the complete report suggests a core course curriculum for computing majors.

The complete report and both appendixes, titled *Report of the ACM Task Force on the Core of Computer Science*, is available for \$7 to ACM members and \$12 to nonmembers by calling ACM at (800) 342-6626 or (301) 528-4261 (Alaska, Maryland, and outside the United States). Specify ACM order number 201880.

findings from scientific experiments. On the theory side, computers help prove theorems, check the consistency of specifications, check for counterexamples, and demonstrate test cases.

Computing sits at the crossroads among the central processes of applied mathematics, science, and engineering. The three processes have equal — and fundamental — importance in the discipline, which uniquely blends theory, abstraction, and design. The binding forces are a common interest in experimentation and design as information transformers, a common interest in computational support of the stages of those processes, and a common interest in efficiency.

Role of programming

We have noted that many activities in computing are not programming, such as hardware design, system architecture, operating system structure, database application design, and model validation. Hence, the notion that “computer science equals programming” is misleading. What role does programming play in the discipline? In the curriculum?

Clearly, programming is part of the standard practices of the discipline and hence every major should achieve competence in it. This does not, however, imply that the curriculum should be based on programming or that the introductory courses should be programming courses.

It is also clear that access to the distinctions of any domain comes through language, and that most of the distinctions of computing are embodied in programming notations. Hence, programming languages are useful tools for gaining access to the distinctions of the discipline.

We recommend, therefore, that programming be a part of the competence sought by the core curriculum, and that programming languages be treated as useful vehicles for gaining access to important distinctions of computing.

Description of computing

Our description of a computing as a discipline consists of four parts:

- requirements,
- short definition,
- division into subareas, and
- elaboration of subareas.

What we say here is merely a snapshot of a changing and dynamic field. We intend

this to be a “living” definition, revised from time to time to reflect maturity and change in the field. We expect revisions to be most common in the details of the subareas, occasional in the list of subareas, and rare in the short definition.

Requirements. There are many possible ways to formulate a definition. We set five requirements for ours:

- (1) It should be understandable by people outside the field.
- (2) It should be a rallying point for people inside the field.
- (3) It should be concrete and specific.
- (4) It should elucidate the historical roots of the discipline in mathematics, logic, and engineering.
- (5) It should set forth the fundamental questions and significant accomplishments in each area of the discipline.

In the process of formulating a description, we considered several previous definitions and concluded that a description meeting these requirements must have several levels of complexity. (See the complete version of this report for the previous definitions.)

Short definition. The discipline of computing is the systematic study of algorithmic processes that describe and transform information, their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, “What can be (efficiently) automated?”

Division into subareas. We grappled at some length with the question of dividing the discipline into subareas. We began with a preference for a small number of subareas, such as model versus implementation, or algorithm versus machine. However, the various candidates we devised were too abstract and the boundaries between divisions were too fuzzy. In addition, we felt that most people would not have identified comfortably with them.

Then we realized that the fundamentals of the discipline are contained in the three basic processes — theory, abstraction, and design — because they are used by the disciplinary subareas to accomplish their goals. Thus, a description of the discipline’s subareas and their relation to these three basic processes would be useful. To qualify as a subarea, a segment of the discipline must satisfy four criteria:

- underlying unity of subject matter,
- substantial theoretical component,
- significant abstractions, and

- important design and implementation issues.

Theory includes the processes for developing the underlying mathematics of the subarea, supported by theory from other areas. For example, the subarea of algorithms and data structures contains complexity theory and is supported by graph theory.

Abstraction deals with modeling potential implementations. These models suppress detail while retaining essential features. They are amenable to analysis and provide means for calculating predictions of the modeled system’s behavior.

Design deals with the process of specifying a problem; transforming the problem statement into a design specification; and repeatedly inventing and investigating alternative solutions until you achieve a reliable, maintainable, documented, tested design that meets your cost criteria.

We considered it desirable that each subarea be identified with a research community, or set of related communities, vital enough to create and maintain a body of literature in the area. We discerned nine subareas that cover the field:

- algorithms and data structures
- programming languages
- architecture
- numerical and symbolic computation
- operating systems
- software methodology and engineering
- database and information retrieval systems
- artificial intelligence and robotics
- human-computer communication

Elaboration of subareas. To present the content of the subareas, we found it useful to think of a 9×3 matrix. Each row is associated with a subarea, with one column each for theory, abstraction, and design. Each square of the matrix holds specific information about that subarea component, describing issues of concern and significant accomplishments.

Certain affinity groups providing scientific literature do not appear as subareas because they are basic concerns throughout the discipline. For example, parallelism surfaces in all subareas (there are parallel algorithms, parallel languages, parallel architectures, and so forth) and in theory, abstraction, and design. Similar conclusions hold for security, reliability, and performance evaluation.

Computer scientists will tend to associate with the first two columns of the matrix, computer engineers with the last two.

The accompanying matrix contains a full description of computing, as specified above.

Curriculum model

Competence in the discipline. The goal of education is to develop competence in a domain, in this case computing. Competence, the capability for effective action, measures individual performance against the standard practices of the field. The criteria are grounded in the history of the field. The educational process that leads to competence includes these elements (see F. Flores and M. Graves, "Education," working paper available from Logonet, 2000 Powell Street, 11th Floor, Emeryville, CA 94608):

- Motivate the domain.
- Demonstrate what can be accomplished in the domain.
- Expose the distinctions of the domain.
- Ground the distinctions in history.
- Practice the distinctions.

This model has interesting implications about the design of a curriculum. The first question it leads to is, "In what areas of computing must majors be competent?" There are two broad areas of competence:

(1) Discipline thinking: the ability to invent new distinctions in the field, leading to new modes of action and new tools that make those distinctions available for others to use.

(2) Tool use: the ability to use the tools of the field for effective action in other domains.

We suggest that discipline thinking is the primary goal of a curriculum for computing majors. Majors must also have a sufficient awareness of the tools so that they can work effectively with those in other disciplines to help design modes of effective action in those disciplines.

The inquiry into competence reveals a number of areas where current core curricula in computing are missing some elements. For example, the historical context of the computing field is often deemphasized, leading to many graduates who are ignorant of history and proceed to repeat its mistakes in their jobs. Many computing graduates wind up in business data processing, a domain in which most computing curricula do not seek to develop competence; it is an old controversy whether computing departments or business departments should develop that competence. Discipline thinking must be based on solid mathematical foundations, yet theory is not

an integral part of most computing curricula. The standard practices of the computing field include setting up and conducting experiments, contributing to team projects, and interacting with other disciplines to support their interests in effective use of computing, yet most curricula neglect laboratory exercises, team projects, or interdisciplinary studies.

The question of the results to be achieved by computing curricula has not been explored thoroughly in past discussions, and we will not attempt a thorough analysis here. We do strongly recommend that this question be among the first taken up in the design of new core curricula for computing.

Lifelong learning. The curriculum should be designed to develop an appreciation for learning that graduates will carry with them throughout their careers. Many courses are oriented around a paradigm that presents "answers" in a lecture format, rather than focusing on the process of questioning that underlies all learning. We recommend that the follow-on committee consider other teaching paradigms that involve processes of inquiry, an orientation to using the computing literature, and the development of a commitment to a lifelong process of learning.

Introductory sequence

In the curriculum model cited above, the motivation and demonstration of the domain must precede instruction and practice in the domain. This is precisely the purpose of the introductory course sequence. The principal areas of computing — in which majors must develop competence — must be shown to the students, with sufficient depth and rigor that students can appreciate the power of the areas and the benefits from achieving competence in them. The remainder of the curriculum must be carefully designed to systematically explore those areas, exposing new concepts and distinctions and giving the students practice in them.

We therefore recommend that the introductory course consist of regular lectures and a closely coordinated weekly laboratory. The lectures should emphasize fundamentals; the laboratories should emphasize technology and know-how.

The recommended model is traditional in the physical sciences and in engineering; lectures emphasize enduring principles and concepts while laboratories emphasize the transient material and skills relating to the

current technology. For example, lectures would discuss the design and analysis of algorithms or the organization of network protocols in functional layers. The corresponding laboratory sessions would require writing programs for algorithms analyzed in lecture and measuring their running times, or installing and testing network interfaces and measuring their packet throughputs.

Within this recommendation, the first courses in computer science would not only introduce programming, algorithms, and data structures, as is now commonly the case, but they would also introduce material from all the other subdisciplines. Mathematics and other theory would be well integrated into the lectures at appropriate points.

We recommend that the introductory course contain a rigorous, challenging survey of the whole discipline. The physics model, exemplified by the Feynman Lectures in Physics, is a paradigm for the introductory course we ultimately envisage.

We emphasize that simply redesigning the introductory course sequence following this recommendation, without redesigning the entire undergraduate curriculum, would be a serious mistake. The experience of physics departments contains many lessons for computing departments in this regard.

Prerequisites. We assume that students who aspire to become computing majors already have a modest background with programming in some language and with computer-based tools such as word processors, spreadsheets, and databases. Given the widening use of computers in high schools and at home, it might seem that universities could assume that most incoming students have such a background and provide a remedial course in programming for the others. We have found, however, that the assumption of adequate high school preparation in programming provokes quite a bit of controversy. Evidence exists that adequate preparation is rare. We therefore recommend that computing departments offer an introduction to programming and computer tools that would be a prerequisite (or corequisite) for the introductory courses. We further recommend that departments provide advanced placement for students with adequate high school preparation.

Formal prerequisites and corequisites in mathematics are more difficult to state and will depend on local circumstances. However, accrediting boards in computing re-

quire considerable mathematics, including discrete mathematics, differential and integral calculus, and probability and statistics. These requirements are often exceeded in the better undergraduate programs. In our description of a beginning computing curriculum, we have spelled out in some detail what mathematics is applicable in each of the nine identified areas of computing. Where possible, we have displayed the required mathematical background for each of the teaching modules we describe. This will allow individual departments to synchronize their own local mathematical requirements and courses with the material in the modules. In some cases it might be appropriate to introduce relevant underlying mathematical topics as needed. In general, we recommend that students see applications of relevant mathematics as early as possible in their computing studies.

Modular organization. The introductory sequence should bring out the underlying unity of the field and should flow from topic to topic in a pedagogically natural way. It would therefore be inadequate to organize the course as a sequence of nine sections, one for each of the subareas; such a mapping would appear as a hodge-podge, with difficult transitions between sections. The following list is an ordering of topics that meets these requirements:

- fundamental algorithm concepts
- computer organization (von Neumann)
- mathematical programming
- data structures and abstraction
- limits of computability
- operating systems and security
- distributed computing and networks
- models in artificial intelligence
- file and database systems
- parallel computation
- a human interface

We have grouped the topics into 11 modules. Each module includes challenging material representative of the subject matter without becoming a superficial survey of every aspect or topic. Each module draws material from several squares of the definition matrix as appropriate. Many modules will therefore not correspond one-to-one with rows of the definition matrix. For example, the first module in our example course is entitled "Fundamental Algorithm Concepts." It covers the role of formalism and theory, method in programming, programming concepts, efficiency, and specific algorithms. It draws information from the first, second, fourth, and sixth rows of the definition matrix. It deals only with sequential algorithms. Later modules,

"Distributed Computing and Networks" and "Parallel Computation," extend the material in the first module and draw new material from the third and fifth rows of the definition matrix.

As a general approach, each module contains lectures that cover the required theory and most abstractions. Theory is generally not introduced until needed. Each module is closely coupled with laboratory sessions, and the nature of the laboratory assignments is included with the module specifications.

The full specification of these modules appears in Appendix II of the complete report. Our specification is drawn up for a three-semester course sequence containing 42 lectures and 35 scheduled laboratory sessions per semester.

We reemphasize that this is intended as an example of a mapping from the disciplinary description to an introductory course sequence. We do not intend this as a prescription for all introductory courses. Other approaches are exemplified by existing introductory curricula at selected colleges and universities.

Laboratories

We have described a curriculum that separates principles from technology while maintaining coherence between the two. We have recommended that lectures deal with principles and laboratories with technology, with the two being closely coordinated. The laboratories serve three purposes:

(1) Laboratories should demonstrate how principles covered in the lectures apply to the design, implementation, and testing of practical software and hardware. They should provide concrete experiences that help students understand abstract concepts. These experiences are essential to sharpen students' intuition about practical computing and to emphasize the intellectual effort in building correct, efficient computer programs and systems.

(2) Laboratories should emphasize processes leading to good computing know-how. They should emphasize programming, not programs; laboratory techniques; understanding of hardware capabilities; correct use of software tools; correct use of documentation; and proper documentation of experiments and projects. Many software tools will be required on host computers to assist in constructing, controlling, and monitoring experiments on attached subsystems; the laboratory should teach proper use of these tools.

(3) Laboratories should provide an introduction to experimental methods, including use of measurement instruments, diagnostic aids, experiment design, software and hardware monitors, statistical analysis of results, and proper presentation of findings. Students should learn to distinguish careful experiments from casual observations.

To meet these goals, laboratory work should be carefully planned and supervised. It is desirable that students attend lab at specified times, nominally three hours weekly. Lab assignments should be pre-planned, with written descriptions of the purposes and methodology of each experiment given to the students. The depth of description should be commensurate with students' prior lab experience: more detail is required in early laboratories. Lab assignments should be carried out under the guidance of a lab instructor who watches that each student follows correct methodology.

The labs associated with the introductory courses will require close supervision and will contain well-planned activities. This implies that more staff will be required per student for these laboratories than for those later in the curriculum.

Lab problems should be coordinated with material in the lecture parts of the course. Individual lab problems in general will deal with combinations of hardware and software. Some lab assignments emphasize technologies and tools that ease the software development process. Others emphasize analysis and measurement of existing software or comparison of known algorithms. Still others emphasize program development based on principles learned in class. (The descriptions of modules in Appendix II contain examples of associated laboratory assignments.)

Laboratory assignments should be self-contained in the sense that the average student should be able to complete the work in the time allocated. Laboratory assignments should encourage students to discover and learn things for themselves. Students should be required to maintain a proper lab book documenting experiments, observations, and data. Students should also be required to maintain their software and to build libraries for use in later lab projects.

We expect that, in labs as in lectures, students will be assigned homework that will require using computers outside the supervised realm of a laboratory. Hence, organized laboratory sessions will supplement, not replace, the usual programming

Definition matrix for computing as a discipline.

	Theory
Algorithms and data structures	Computability theory; computational complexity theory; algorithmic time and space bounds; levels of intractability; parallel computation, lower bounds, mappings from algorithms' dataflow requirements into machine communication paths; probabilistic algorithms; cryptography; supporting areas of graph theory, recursive functions, recurrence relations, combinatorics, calculus, induction, predicate and temporal logic, semantics, probability, and statistics
Programming languages	Formal languages and automata; Turing machines, Post Systems, λ -calculus; formal semantics; supporting areas of predicate logic, temporal logic, modern algebra, and mathematical induction
Architecture	Boolean algebra; switching theory; coding theory; finite state machine theory; supporting areas of statistics, probability, queueing, reliability theory, discrete mathematics, number theory, and arithmetic in different number systems
Numerical and symbolic computation	Number theory; linear algebra; numerical analysis; nonlinear dynamics; supporting areas of calculus, real analysis, complex analysis, and algebra
Operating systems	Concurrency theory; scheduling theory; program behavior and memory management theory; performance modeling and analysis; supporting areas of bin packing, probability, queueing theory, queueing networks, communication and information theory, temporal logic, and cryptography
Software methodology and engineering	Program verification and proof; temporal logic; reliability theory; supporting areas of predicate calculus, axiomatic semantics, and cognitive psychology
Databases and information retrieval	Relational algebra and relational calculus; dependency theory; concurrency theory; statistical inference; sorting and searching; performance analysis; supporting area of cryptography
Artificial intelligence and robotics	Logic; conceptual dependency; cognition; syntactic and semantic models for natural language understanding; kinematics and dynamics of robot motion and world models used by robots; supporting areas of structural mechanics, graph theory, formal grammars, linguistics, philosophy, and psychology
Human-computer communication	Geometry of two and higher dimensions; color theory; cognitive psychology; supporting areas of Fourier analysis, linear algebra, graph theory, automata, physics, and analysis

Abstraction

Efficient, optimal algorithms and analyses for best, worst, and average performance; classifications of the effects of control and data structure on time and space requirements; important classes of techniques; parallel and distributed algorithms, methods of partitioning problems into tasks executable in separate processors

Classification of languages based on their syntactic and dynamic semantic models; classification of languages by intended application area; classification of major syntactic and semantic models for program structure; abstract implementation models for major types of language; methods for parsing, compiling, interpretation, and code optimization; methods for automatic generation of parsers, scanners, compiler components, and compilers

Finite state machine and Boolean algebraic models of circuits relating function to behavior; other general methods of synthesizing systems from basic components; models of circuits and finite state machines for computing arithmetic functions over finite fields; models for data path and control structures; optimizing instruction sets for various models and workloads; hardware reliability; space, time, and organizational trade-offs in design of VLSI devices; organization of machines for various computational models; identification of design levels

Formulations of physical problems as models in continuous (and sometimes discrete) mathematics; discrete approximations to continuous problems; finite element model for problems specifiable by regular meshes and boundary values, associated iterative methods and convergence theory, parallel solution methods, automatic grid refinement during numerical integration; symbolic integration and differentiation

Abstraction principles that permit user operation on idealized versions of resources; binding of objects perceived at the user interface to internal computational structures; models for important subproblems; models for distributed computation; models for secure computing; networking

Specification methods; methodologies; methods for automating program development; methodologies for dependable computing; software tools and programming environments; measurement and evaluation of programs and systems; matching problem domains through software systems to particular machine architectures; life-cycle models of software projects

Models for representing logical structure of data and relations among data elements; representations of files for fast retrieval; methods for assuring integrity (consistency) of database under updates; methods for preventing unauthorized disclosure or alteration and for minimizing statistical inference; languages for posing queries over databases of different kinds, similarly for information retrieval systems; models that allow documents to contain text at multiple levels, plus video, graphics, and voice; human factors and interface issues

Knowledge representation and methods of processing them; models of natural language understanding and representations, machine translation; speech recognition and synthesis, translation of text to speech; reasoning and learning models; heuristic search methods, branch and bound, control search; machine architectures that imitate biological systems; models of human memory, autonomous learning, and other elements of robot systems

Algorithms for displaying pictures; models for CAD; computer representations of physical objects; image processing and enhancement methods; man-machine communication

Design

Selection, implementation, and testing of algorithms; implementation and testing of general methods, distributed algorithms, and storage managers; experimental testing of heuristic algorithms for combinatorial problems; cryptographic protocols

Specific languages that combine an abstract machine (semantics) and syntax to form a coherent, implementable, whole; specific implementation methods for particular classes of languages; programming environments; parser and scanner generators; programs for syntactic and semantic error checking, profiling, debugging, and tracing; applications of programming language methods to document-processing functions, other applications

Hardware units for fast computation; von Neumann machine: single-instruction sequence stored program computer, RISC and CISC implementations; efficient methods of storing and recording information and detecting and correcting errors; specific approaches for responding to errors; CAD systems and logic simulations for design of VLSI circuits, production programs for layout and fault diagnosis, silicon compilers; implementing machines in various computational models; supercomputers

High-level problem formulation systems; specific libraries and packages for linear algebra, ordinary differential equations, statistics, nonlinear equations, and optimizations; methods of mapping finite element algorithms to specific architectures; symbolic manipulators

Prototypes of time-sharing systems, automatic storage allocators, multi-level schedulers, memory managers, hierarchical file systems, and other important system components that have served as bases for commercial systems; techniques for building operating systems; techniques for libraries of utilities; files and file systems; queueing network modeling and simulation packages for evaluating performance of real systems; network architectures; protocol techniques embodied in TCP/IP, virtual circuit protocols, internet, real-time conferencing, and X.25

Specification languages, configuration management systems, revision control systems; syntax-directed editors, line editors, screen editors, word processing systems; specific methodologies advocated and used for software development; procedures and practices for testing, quality assurance, and project management; software tools for program development and debugging, profiling, text formatting, and database manipulation; specification of criteria levels and validation procedures for secure computing systems; design of user interfaces; methods for designing very large computer systems that are reliable, fault-tolerant, and dependable

Techniques for designing databases for relational, hierarchical, network, and distributed implementations; techniques for designing database systems; techniques for designing information retrieval systems; design of secure database systems; hypertext systems; techniques to map large databases to magnetic disk stores; techniques for mapping large, read-only databases onto optical storage media

Techniques for designing software systems for logic programming, theorem proving, and rule evaluation; techniques for expert systems in narrow domains and expert system shells programmable for new domains; implementations of logic programming; natural language understanding systems; implementations of neural networks and sparse distributed memories; programs that play games of strategy; working speech synthesizers, recognizers; working robotic machines, static and mobile

Implementation of graphics algorithms on various graphics devices; design and implementation of experimental graphics algorithms; proper use of color graphics for displays, accurate reproduction of colors; graphics standards, languages, and graphics packages; implementation of various user interface techniques; implementation of various standard file interchange formats; working CAD systems; working image enhancement systems

and other written assignments.

In a substantial number of labs dealing with the development of programs, the assignment should be to modify or complete an existing program supplied by the instructor. This forces the student to read well-written programs, provides experience with integration of software, and results in a larger and more satisfying program for the student.

Computing technology constantly changes. It is difficult, therefore, to give a detailed specification of the hardware systems, software systems, instruments, and tools that ought to be in a laboratory. The choice of equipment and staffing in laboratories should be guided by the following principles:

(1) Laboratories should be equipped with up-to-date systems and languages. Programming languages have a significant effect on shaping a student's view of computing. Laboratories should deploy systems that encourage good habits in students; it is especially important to avoid outdated systems (hardware and software) in core courses.

(2) Hardware and software must be fully maintained. Malfunctioning equipment will frustrate students and interfere with learning. Appropriate staff must be available for maintaining the hardware and software used in the lab. The situation is analogous to laboratories in other disciplines.

(3) Full functionality is important. (This includes adequate response time on shared systems.) Restricting students to small subsets of a language or system might be useful in initial contacts, but the restrictions should be lifted as the students progress.

(4) Good programming tools are needed. Compilers get a lot of attention, but other programming tools are used as often. In Unix systems, for example, students should use editors like Emacs and learn to use tools like the shell, grep, awk, and make. Storage and processing facilities must be sufficient to make such tools available for use in the lab.

(5) Adequate support for hardware and instrumentation must be provided. Some projects might require students to connect hardware units together, take measurements of signals, monitor data paths, and the like. A sufficient supply of small parts, connectors, cables, monitoring devices, and test instruments must be available.

The IEEE Computer Society Task Force on Goal-Oriented Laboratory Development has studied this subject in depth. Its report includes a discussion of the resources — staff and facilities — needed for laboratories at all levels of the curriculum.

Accreditation

We conducted this work with the intent that example courses be consistent with current guidelines of the Computing Sciences Accreditation Board. Nonetheless, the details of the mapping of this content to CSAB guidelines does not come within the purview of this committee.

We designed this report to provoke new thinking about computing as a discipline by exhibiting the discipline's content in a way that emphasizes fundamental concepts, principles, and distinctions. The report also suggests a redesign of the core curriculum

according to an education model used in other disciplines: a progression from demonstrating the existence of useful distinctions to practice that develops competence with them. We illustrated that by proposing a rigorous introductory course that puts the concepts and principles into the lectures and the use of technology into closely coordinated laboratories. A department cannot simply replace its current introductory sequence with the new one; it must redesign the curriculum so that the new introduction is part of a coherent whole. For this reason, we recommend that the ACM establish a follow-on committee to complete the redesign of the core curriculum.

We recognize that many practical problems must be dealt with before a new curriculum model can become part of the field. For example,

(1) Several years will be required for faculties to redesign their curricula based on a new conceptual formulation.

(2) Currently, no textbooks or educational materials are based on the framework proposed here.

(3) Most departments have inadequate laboratories, facilities, and materials for the educational task suggested here.

(4) Teaching assistants and faculty are inexperienced in the new approach.

(5) Good high school preparation in computing is rare.

We recognize that many of the recommendations given here are challenging and will require substantial work to implement. We are convinced that the improvements in computing education from the proposals here are worth the effort, and we invite you to join us in achieving them. □

Acknowledgments

A large number of people generously provided written comments in response to drafts of this report. Although it was not possible to accommodate every comment in detail, we did take every comment into account in finalizing this report. We are grateful to the following people for sending us their comments: Paul Abrahams, J. Mack Adams, Robert Aiken, Donald Bagert, Alan Biermann, Frank Boesch, Richard Botting, Albert Briggs, Jr., Judy Brown, Rick Carlson, Thomas Cheatham, Neal Coulter, Steve Cunningham, Verlynda Dobbs, Caroline Eastman, Richard Epstein, Frank Friedman, C.W. Gear, Robert Glass, Nico Habermann, Judy Hankins, Charles Kelemen, Elliot Koffma, Barry Kurtz, Doris Lidtke, Michael Loui, Paul Luker, Susan Merriitt, John Motil, J. Paul Myers, Ken Nennedy, Bob Noonan, Alan Perlis, Jesse Poore, Terrence Pratt, Jean Sammet, Mary Shaw, J.W. Smith, Dennis Smolarski, Ed Upchurch, Garret White, Gio Wiederhold, and Stuart Zweben.

Moving?

PLEASE NOTIFY
US 4 WEEKS
IN ADVANCE

Name (Please Print)

New Address

City

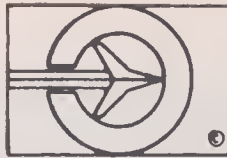
State/Country

Zip

MAIL TO:
IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

ATTACH
LABEL
HERE

- This notice of address change will apply to all IEEE publications to which you subscribe.
- List new address above.
- If you have a question about your subscription, place label here and clip this form to your letter.



Call for Papers

Fifth Aerospace Computer Security Applications Conference

December 4-8, 1989
Westward Look Hotel
Tuscon, Arizona

The Conference

Operational requirements for civil and military systems increasingly stress the necessity for information to be readily accessible. The Computer Security Act of 1987 requires that all Federal agencies take certain actions to improve the security and privacy provided by federal computer systems. Accomplishing both operational and security requirements requires the application of the maturing technology of computer security to new and existing systems throughout their life cycle.

This conference will explore technology applications in complementary aspects: the policy issues and operational requirements for both civil and military systems; the hardware and software tools and techniques being developed to satisfy system requirements; and specific examples of systems applications and implementations.

Papers and Tutorials

Technical papers and tutorials which address the application of computer security technologies in the aerospace and other environments are solicited. Original research, analyses and approaches for defining the computer security issues and problems identified in the Conference's interest areas; secure systems in use or development; methodological approaches for analyzing the scope and nature of computer security issues; and potential solutions are of particular interest. Full papers must be mailed before 19 May, 1989. Material should be sent to:

Dr. Ronald A. Gove
Technical Program Chair
Booz-Allen & Hamilton Inc.
4330 East-West Highway
Bethesda, MD 20814
(301) 951-2395
Gove@dockmaster.arpa

Dr. Dixie B. Baker
Tutorial Program Chair
The Aerospace Corporation
P.O. Box 92957
2350 East El Segundo Blvd.
El Segundo, CA 90245-4691
(213) 336-3049
baker@aerospace.arpa

Robert D. Kovach
Exhibits Chair
ACSA-EX
6946 33rd Street N.W.
Washington, D.C. 20015
(202) 453-1182
rkovach%nasamail@ames.arpa

Areas of Interest Include:

ISO/OSI Security Architecture
GOSIP
SDNS
Trusted DBMSs and Operating Systems
Network Security
Current and Future Trusted System Technology
Space Station Requirements

Certification, Evaluation,
and Accreditation
Policy and Management Issues
Advanced Architectures
C³I Systems
Risk/Threat Assessments
Public Law 100-235

Additional Information

For more information or to receive future mailings, please contact the following at:

Marshall Abrams
Conference Chairman
(703) 883-6938
abrams@mitre.org

Diana Akers or Victoria Ashby
Publicity Chair and Co-Chair
(703) 883-5907 or (703) 883-6368

The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102



IEEE COMPUTER SOCIETY



THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.

1988 Snowbird Report: A Discipline Matures

David Gries, Chair, Computing Research Board

Terry Walker, Chair, Snowbird 88

Paul Young, Vice Chair, Computing Research Board

Chairs of departments in the United States and Canada that offer PhDs in computer science and engineering had their biannual meeting in Snowbird, Utah, July 11-13, 1988. The meeting was attended by representatives of 105 of the 150-odd departments and 27 participants from government and industry. The meeting was organized by the Computing Research Board, an incorporated board of 24 directors elected by the PhD-granting computing departments, whose goal essentially is to represent the research and academic interests of computing (i.e., computer science and engineering).

At Snowbird 88, the impression emerged that the discipline of computing was maturing and coming into balance. Many of the problems that plagued the discipline in the late 1970s and 1980s had been solved or alleviated. It was time for the discipline to cease its largely inward-looking activities and branch outward. As an enabling technology for other disciplines, computing should take a more active role in articulating how its own needs, concerns, and basic research impact other disciplines and in collaborating with other disciplines in the evolution of computing applications. Furthermore, the consensus at Snowbird 88 was that the Computing Research Board should take aggressive steps to articulate and represent the needs and opportunities of computing research.

**No single entity
represents computing
research to the public
and to policy makers.**

**As a result of
Snowbird 88, the
Computing Research
Board plans to fill
that role.**

The end of a crisis

The late 1970s and early 1980s were trying times for computer science departments. The 1980 Snowbird Report, "A Discipline in Crisis,"¹ compared the doubling of course enrollments from 1975 to 1979 with the three-percent gain in faculty in PhD-granting departments over the same four-year period. It discussed the woefully inadequate experimental computing facilities and severe space shortages of most departments. It also suggested steps toward solving the crisis.

The crisis still remained in 1982, according to that year's report, "Meeting the Crisis in Computer Science."² But there were signs of improvement. The report outlined further steps that universities, government, and industry should take to bring relief.

By Snowbird 84, computer science was out of the critical stage, but still in need of substantial help.³ That meeting, devoted to an inward look at the field, dealt with issues such as the content of computer science, departmental administration, the youth of the field and its attendant problems, departmental infrastructure requirements, and long-range planning.

Snowbird 86 also dealt mainly with internal issues. Perhaps the most important discussions centered around the imbalance between the by-then-apparent growth in personnel in the field and the inadequate growth in funding of basic research.⁴

Snowbird 88 centered around the feeling that the field of computing has in some sense "come of age." Internal problems of the late 1970s and early 1980s have been resolved in some cases and become more manageable in others. And, in some arenas, the field has achieved a stature equivalent to that of other scientific fields. For example,

- PhD production for 1987-88 was estimated at 500, more than double that of 10 years earlier.⁵ With over 50 percent of the

PhDs taking faculty positions, the critical shortage of faculty members is beginning to ease in PhD-granting departments.

- A report⁶ by Richards Adrion of the University of Massachusetts and Paul Young of the University of Washington indicated that the critical space shortages for PhD-granting computing departments are being overcome: 78 departments responding to a survey have gained or will gain 27 new buildings and 50 substantial renovations in the 1985-1991 period.

- While the three-year trend of decreasing enrollments in computer science has caused concern, it has also brought relief. The sense of being overwhelmed by enormous student-faculty ratios is gone.

- With more assistant professors than full professors, the field is still plagued — administratively and politically — by its youth. But the field is maturing, and more of its scientists are well-known, respected, and willing to give their time in serving the field.

- The National Science Foundation has established the Computer and Information Science Directorate of the NSF, first under Gordon Bell and now under Bill Wulf. This places computing — that is, computer science and engineering — on a more equal footing with other sciences.

- The National Research Council has established a Computer Science and Technology Board, with Joe Traub as its first chair. As Traub reported at Snowbird 88, the CSTB interacts with policy makers, monitors the field, and is responsible for studies on various aspects of computing. This move, too, places computing on a more equal footing with other scientific and engineering disciplines.

Discussions at Snowbird 88 also indicated that the field of computing has matured to the point that it can now begin to reach outward. Now that we have handled the severe internal problems and have trained a new generation of responsible, experienced people, there is time to devote to more outward-looking activities. This opinion was reflected in discussions on research directions, on professional responsibilities, and on the view the field presents to the outside world.

Articulating the needs and possibilities of computing research

Many Snowbird 88 speakers stressed the need for the field to be more articulate about, and to pay more attention to, what

computing can do as an enabling discipline for other fields. Making this point were Les Belady of MCC, Mike Dertouzos of MIT, Joe Traub of CSTB, and Peter Freeman and Bill Wulf of the NSF. Freeman and Wulf, in particular, spoke of the need for articulate, recognized representation as the field speaks to other disciplines and to national policy makers.

Many speakers criticized research efforts as too exclusively inward-directed. Dertouzos, for example, urged more collaboration with researchers in other fields. "An architect does not throw bricks at a client and tell him to build a building," he said. "He gets involved and tailors his aesthetic and technical skills to the client's needs."

In the past, the field has often failed to show how basic computing research impacts the national scientific and engineering effort. As the 1990s draw near, it must be more aggressive about articulating important research directions. It must also deal more successfully with the image it presents to outsiders and its own constituency — that was perhaps the most important topic discussed at Snowbird 88.

The outside image of computing research. In the past, no group has actively and steadily represented computing research interests, and no group has been consistently sought by outsiders for information, insight, and policy. Each of the other major scientific disciplines (for example, mathematics and physics) has a well-established, respected professional society that speaks for its researchers and represents the discipline in the eyes of Congress, other federal agencies, foundations, the press, and the public. This society is the primary source for information, general explanations of technical results, names of people to serve on committees and boards, and help in formulating policy. The society aggressively pursues its mission, actively taking its information to those

who need it and not just passively waiting for requests.

Such activity helps maintain and increase funding for the field, but it has the more important goal of educating others. Decisions on policy concerning science, research, technology, and the economy should be made with as much information as possible. A field must make a commitment to perform this education, and in a consistent fashion, for no one else will.

Those who set policy and establish priorities have trouble hearing those who remain silent. Consider, for example, the case of mathematics. Throughout the 1970s, math was quiet about its needs and its importance, and it suffered because of it. Several years ago, the American Mathematics Society (AMS), the Mathematical Association of America (MAA), and the Society for Industrial and Applied Mathematics (SIAM) joined forces, sponsored the influential David Report, and formed the Joint Policy Board for Mathematics, supported by a full-time staff of three in Washington, DC. The JPBM, which is responsible for most of the recent publicity for mathematics, has had a dramatic effect. Policy makers are now aware of the role mathematics plays and of its plight in the past decade.

Another example concerns the Computing Research Board. During their work on the Trends Report,⁴ the authors received a number of phone calls from people and agencies, requesting explanations of the board's activities and goals. Subsequently, the National Research Council reformed the Computer Science and Technology Board, which had languished for years. We believe that the publicity generated by the Trends report is partially responsible for CSTB's resurrection and computer science and engineering's increased visibility within NSF.

Peter Freeman of the NSF summed up the problem of computing research advocacy nicely: "From my perspective at

Post-Snowbird accomplishments

Since July 1988, the Computing Research Board has been working to establish itself as a representative for computing research. PhD-granting departments now pay dues, and other fund-raising activities are underway. The board has appointed Terry Walker as the full-time executive director. Discussions have been held with the Computer Society, ACM, and SIAM. As a result, the Computer Society has provided a Washington, DC, office for Walker, the board has joined ACM in planning a conference on research directives, and SIAM is assisting in the development of a newsletter.

the NSF, it is not that we won't come to an outside organization, but that at present there is no one to speak for the research community in the same way there is in physics, math, and chemistry. There are two issues: physical proximity and legitimacy. At present, no one can claim to speak for the whole computer science research community; policy makers don't recognize any organization as the spokesman."

Also, in his banquet speech, Wulf talked about the need for a consistent, clearly recognized group to articulate the field's research needs and important achievements.

The internal image of a field. Besides keeping the public informed, a professional society keeps its own membership informed about issues, policies, opportunities (prizes, awards, fellowships, funding programs, etc.), and responsibilities. A society keeps its research community aware of what the field is doing and what others are doing for the field. It strives to broaden the outlook of its members. It promotes awareness that research achievements occur within the context of the field as a whole and that resources for research are often built on a political base. Researchers develop a sense of belonging and professional responsibility.

This is done partially through well-attended national meetings and professional newsletters. Examples of good newsletters are *SIAM News* and *Notices of the American Mathematical Society*, which are full of news from Washington, short technical overviews of interesting subfields, news briefs, opportunities (grant and fellowship deadlines, awards, prizes, committee work, etc.), information about people, and commentary on educational issues. In computing, *Communications of the ACM* and *Computer* do provide some channels for disseminating such material, but they have not been used much in the past and are not considered important conveyors of this material.

The youth of computer science has made it particularly difficult for us in this regard. With rapid growth and change, with more assistant professors than full professors (a problem computing alone "enjoys"), our senior people often have excessive administrative duties in their home institutions and little time for outside duties. Further, younger people come in contact with few scientists who are engaged in promoting the interests of the field; their role models are engaged more

The computing field is not educating its younger people about their professional responsibilities.

in research and university activities. As a result, the field is not educating its younger people about their professional responsibilities the way other fields do.

Symptoms of the lack of awareness of professional responsibility abound. Editors complain about the dearth of good referees. Committees like the ACM Turing Award Committee receive few well-prepared nominations. Recruitment of new members for the ACM and the Computer Society is shockingly small, partially because the field is not encouraging its members to join. Researchers avoid leadership positions in societies, or don't even join societies. Throughout the 1980s, the Computer Science Division within NSF has had difficulty finding rotating program directors. And newer and smaller PhD-granting departments that consist mainly of young people do not fully understand the funding process and have difficulty attracting research funds.

Computing research lacks representation. The computing research community lacks a single entity that represents its research interests, both to the public and to its constituents. From time to time, the Computing Research Board has responded to crises, but it lacks the funds and personnel to perform consistently in the periods between.

Partly because their constituencies are much broader, the ACM and the Computer Society have not fulfilled this role for the research community. They both have excellent technical journals and sponsor scientific conferences, so they perform some of the functions of a research society quite admirably. But they do not perform the functions discussed above.

The lack of a single entity to represent computing research has negative effects. For example,

- From 1979 to 1985, both total-federal and NSF-specific constant-dollar funding

of basic research in computer science failed to grow as rapidly as the number of academic personnel; in all other related major fields, constant-dollar funding grew more rapidly. From 1979 to 1985, NSF constant-dollar funds for individual grants in computer science declined by eight percent, even though the number of researchers in the field grew substantially. (This resulted partially from the introduction and growth of the Coordinated Experimental Research program in computer science to introduce infrastructure and research facilities similar to those already existing in many other disciplines.) Substantial growth in personnel did not entitle computing research to more funds, and the intellectual case for more funds was not aggressively and consistently pursued.

- Several years ago, congressional staff and other policy makers mistakenly believed they had given huge sums to computer science in setting up the four NSF supercomputer centers, which were being used mainly by other scientists. Like many people, they associated the funding of computers with the funding of computer science and engineering.

- The Department of Education just developed a program to provide graduate fellowships in scientific and engineering areas of national need. Computer science was not included because it was not considered an area of national need; physics, chemistry, mathematics, and all the engineering disciplines were included.

- The American Association for the Advancement of Science has a project, called Project 2061, to deal with education for a changing future. The draft of a report, "What Science is Most Worth Knowing," was written by some 42 scientists, eight of whom were mathematicians. The academic computer science community had essentially no representation.

- A May 20, 1988, *Science* editorial called for the creation of a Panel for Science Priorities because "the paths of Big Science and big federal deficits have finally supercollided." The panel members would be selected by the appropriate societies. Were such a panel chosen today, it is safe to predict that computing would again be represented by those outside the field.

A major change in activities

Speaking for the Computing Research Board at Snowbird 88, David Gries dis-

Policy makers associated the funding of computers with the funding of computer science and engineering.

cussed the problem of constructing a single voice for computing research in today's context, where the ACM, the Computer Society, and SIAM already exist. Would it be possible to form a coalition of 150 academic departments, 50 industrial labs, and three societies with over 160,000 members? Such a coalition could have a far greater impact on the image of computing in Washington and with the public than any society could have on its own, and each society's contribution would be relatively modest compared to the substantial expenditure needed.

Gries proposed that the Computing Research Board head that coalition. In anticipation of such a move, the board has already incorporated. It has also changed its mode of election and is now elected by the chairs of PhD-granting computing departments.

The Snowbird participants felt that the Computing Research Board should hire a permanent staff, whose duties would include the following:

(1) Develop a newsletter, initially to be distributed free to all faculty members in the PhD-granting computing departments (at the least). The goal is to make computer scientists and engineers more aware of issues that affect computing research and to make them more conscious of their professional duties. It was suggested that *SIAM News* would be an excellent model to follow in terms of frequency and content.

(2) Establish a computing research presence in Washington. The Computing Research Board would seek the endorsement and support of ACM, the Computer Society, and SIAM as the single entity representing computing research interests. The three societies would contract with the board to perform this service for the entire computing community.

This action, which would require sub-

stantial funds, would be a major change for the Computing Research Board. Founded in 1973 to represent the interests of the PhD-granting computer science departments, the board has functioned since then with almost no income. Members (or their institutions) have paid their own travel expenses to meetings. The Taulbee survey and the Forsythe list of PhD-granting departments are subsidized by members' institutions, and there have been no funds for promoting the interests of the research community.

The representatives of the 105 computing departments attending Snowbird 88 expressed full support for this plan. They felt that the Computing Research Board should do it right or not at all and concluded that the cost would be \$400,000 to \$500,000 per year. Most departments represented at Snowbird 88 were willing to pay substantial dues to support the plan — from \$500 to \$5,000 per department per year.

At its meeting directly following Snowbird 88, the board accepted this mandate to move aggressively to represent the interests of the computing field. It is now proceeding on three fronts: developing a set of tasks for a permanent staff, developing a dues structure and other funding sources, and initiating discussions with ACM, the Computer Society, and SIAM.

References

1. P.J. Denning (ed.), "The 1980 Snowbird Report: A Discipline in Crisis," *Comm. ACM*, Vol. 24, No. 6, June 1981, pp. 370-374.
2. S.S. Yau (ed.), "The 1982 Snowbird Report: Meeting the Crisis in Computer Science," *Comm. ACM*, Vol. 26, No. 12, Dec. 1983, pp. 1046-1050.
3. J. Tartar (ed.), "The 1984 Snowbird Report: Future Issues in Computer Science," *Comm. ACM*, Vol. 28, No. 5, May 1985, pp. 490-493.
4. D. Gries et al., "Imbalance Between Growth and Funding in Academic Computing Science: Two Trends Colliding," *Computer*, Vol. 19, No. 9, Sept. 1986, pp. 70-76, and *Comm. ACM*, Vol. 29, No. 9, Sept. 1986, pp. 870-878.
5. D. Gries and D. Marsh, "The 1986-87 Taulbee Survey Report," *Comm. ACM*, Vol. 30, No. 8, Aug. 1988, pp. 984-991, and *Computer*, Vol. 21, No. 8, Aug. 1988, pp. 53-61.
6. W.R. Adrion and P. Young, "Space in Time: A Report on Trends in Space Available for Departments of Computer Science and Computer Engineering," June 1988. (Available from the Computing Research Board, 1730 Massachusetts Ave., NW, Washington, DC, 20036-1903.

Late Magazines?
No Magazines?
Membership
Status Problems?
No Answers
To Your
Complaints?

Let your
Computer
Society
Ombudsman
cut
through
the red
tape
for you.



THE COMPUTER SOCIETY
Ombudsman
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
COMPMAIL+: CS.HELP



"MEMBERSHIP
IS PART
OF
BEING A
PROFESSIONAL."

JOIN TODAY!

Use the application on page
113 or for more information,
circle number 202
on the reader service card.

THE COMPUTER SOCIETY
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-2578
Telephone: (714) 821-8380

Discontinuities in computer education

*Kenneth Magel,
North Dakota State University*

Some computer science undergraduate programs differ in philosophy and some in history, but most differ in the amount of financial support they receive. Differences due to financial support especially are growing rapidly. Is the computer science education at the well-supported schools significantly and qualitatively different from that at other schools? Should only the hundred or so well-supported schools teach computer science while some two thousand other schools change their programs to data processing?

We can examine this issue more closely by characterizing computer science according to at least three criteria: degree of formalism, student assistance, and equipment.

Formalism. Programs range from data processing programs that emphasize programming so strongly that consideration of principles is almost eliminated, through programs based on the traditional college mathematics model, to programs that emphasize principles from the beginning. In the math model, the student first learns how to use the tools (calculus in the case of mathematics, programming languages in the case of computer science) and then learns the theory behind the tools (analysis in math, proofs of correctness and models of computation in computer science). Generally, though, programs have been moving away from tool-using and toward the teaching of general principles.

This trend leads to three major problems:

(1) Companies, especially smaller companies, find that their new hires need more on-the-job training to become productive and are less satisfied with their jobs.

(2) The faculty at smaller colleges often lacks deep understanding of theoretical foundations. Theory-based textbooks are not enough.

(3) Motivating good students to complete the first year of a theory-based program, particularly when many of them are already accomplished microcomputer programmers, is extremely difficult.

Student assistance. Very small programs at undergraduate-oriented colleges often require that each professor spend considerable time assisting individual students. On the other hand, well-supported programs at research-oriented schools can provide teaching assistants to handle student questions and problems. Unfortunately, an increasing number of programs have many more students than faculty to assist them, and they are not funded well enough to provide adequate numbers of teaching assistants. Even though the number of computer science majors has dropped substantially since 1982, severe funding difficulties in many states have worsened the problem.

Equipment. Well-supported schools can provide workstations, possibly

networked to minicomputer servers. Many other schools are limited to an inadequate number of IBM PC-class microcomputers. Still other schools use only a larger computer on a time-sharing basis. As software such as OS/2 or CASE tools becomes more prevalent and more expensive, the chasm will widen between schools that can provide modern software and those that cannot. Large computer companies make the problem worse by concentrating their grants and gifts on the most prestigious schools, even though most of their employees come from the other institutions.

Educators and industry people must decide if the emerging qualitative diversity among computer science programs is beneficial. If it is not, they must take action and rethink their funding patterns.

What is a pack computer?

Stanley Lass, consultant

Accessing multiple words per cache access can ease the cache bottleneck in current computers. With this approach, the compiler must group (or package) both instructions and data to minimize the useful words per access. The packaged instructions and data are called packs, hence the name "pack computer."

A pack computer could have a rather small program memory and data memory (64 words for each). The computer must continue reading instruction packs into the program memory, then executing them. An instruction pack usually loads its successor pack. Data is read and written as packs.

For a small program memory, the simple way to package instructions involves packaging them from a branch target up to and including the next branch instruction. A more complex scheme packages complete loops or short if-then-else-end structures within packs; the computer could perform the intrapack branches. Most complex is a system in which the program memory holds many packs and the computer performs interpack branches, including function

calls and returns. Optimizing use of program memory would be similar to optimizing register use.

Data packs can be vectors, files in memory, records, a group of local variables, a group of parameters, and individual variables. Long packs would be double buffered.

Since memory requests are grouped into pack requests, a memory system operates more efficiently and can support a faster execution rate, either within a single processor or in multiple processors.

Dual threading permits computation on one thread while the other waits for a pack request to complete.

Since a pack computer makes many fewer memory requests per instruction executed, the computer and its memory system are less closely coupled, so the design of each can be optimized.

Hardware caches would need to support pack requests.

Algorithm design should minimize accesses.

With the coming faster clock speeds and parallel execution of instructions, pack computers could provide a marked improvement in cache performance.



IEEE COMPUTER SOCIETY



THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.

IEEE
COMPUTER
SOCIETY
PRESS

Presenting the NEWEST RELEASES

from the Computer Society Press

COMING SOON!

Monograph: Analyzing Computer Architectures
by Jerome C. Huck & Michael J. Flynn

Compares architectural features of instruction sets by measuring program execution under an identical work load. Introduces the mechanisms used to measure high-level language programs and instruction set execution. Presents the results of measuring the four architectures and examines the variations in the instruction stream due to the compiler. **Casebound.**

CONTENTS: Introduction; Program and Instruction Set Measurement; Comparative Data; Compiler Effects on Instruction Execution; Varying Architectures.

Order No. FQ857

ISBN 0-8186-8857-2; May 1989, c. 200 pp.

list price \$36.95/member price \$24.95 (pre-publication price)

JUST RELEASED!



NEW!

Tutorial: Parallel Architectures for Database Systems

by Hurson, Miller, & Pakzad

Focuses on the architectural solutions to the database problem--namely, database machines. Provides insight on the past, present, and future of database machines. Overviews some of the early designs as well as several current designs and commercial systems.

CONTENTS: Introduction to Database Machines; Theory of Relational Data Model; Classification and Architecture of Database Machines; Join Operation; New Directions: Dataflow Database Machines, Main Memory and Temporal Databases, and Null Values.

Order No. FQ838

ISBN 0-8186-8838-6

Jan. 1989, 486 pp.

list price \$59.95

member price \$40.95

NEW!

Tutorial: Distributed-Software Engineering

by Sol Shatz & Jia-Ping Wang

Considers each of the key problems involved in distributed-software analysis, design, implementation, and testing. Focuses on developing correct and reliable software, the most significant problem in developing distributed computing systems.

CONTENTS: Introduction; Requirements for Distributed Software; Design of Distributed Software: Task Partitioning and Allocation; Distributed Programming; Testing and Distributed Software: Dynamic Testing, Debugging, and Static Analysis; Bibliography; Indices.

Order No. FQ856

ISBN 0-8186-8856-0

Jan. 1989, 294 pp.

list price \$44.95

member price \$30.95

NEW!

Tutorial: Microprogramming and Firmware Engineering

by V. Milutinović

Discusses various issues in microprogramming and firmware engineering. Concentrates on the fundamentals of the field, not on the most recent advances.

CONTENTS: Hardware and Architecture Issues; Optimization Concepts; Horizontal Microcode Compaction--Local and Global Methods; Minimization of the Microcode Width; Emulation and HLL Machines; Vertical Migration; Microcode Development; Testing and Verification; Low and High-Level Languages for Microprogramming; Applications.

Order No. FQ839

ISBN 0-8186-0839-0

Jan. 1989, 418 pp.

list price \$49.00

member price \$36.00

ORDER TODAY!

Charge It!

Call toll-free 1-800-CSBOOKS (1-800-272-6627) (in California call 714-821-8380) 8am to 4pm Pacific Time

We accept: VISA • MasterCard • American Express

Please have your IEEE or CS membership number available for member discount.

(Prices outside North America vary. All books subject to availability and prices subject to change.)

Or Write:

U.S. & Canada: Customer Service, IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720 USA

Europe: Customer Service, IEEE Computer Society, 13, Avenue de l'Aquilon, B-1200 Brussels, BELGIUM

Asia & Australia: Customer Service, IEEE Computer Society, Ooshima Building, 2-19-1 Aoyama, Minato-ku, Tokyo 107 JAPAN

Interested in authoring a CS Press book?

Contact: Editor-in-Chief, CS Press, 1730 Massachusetts Avenue, N.W., Washington, DC 20036

STANDARDS

Editor: Helen M. Wood, National Oceanic and Atmospheric Administration, Rm. 1069, Federal Bldg. No. 4, Washington, DC 20233, phone (301) 763-1564.

Rugged Bus, Futurebus groups to cooperate in producing key standards

Two IEEE Computer Society organizations have announced that they intend to join forces to create one set of computer bus standards that will become the basis for a new generation of open-system computers. The organizations are the IEEE P896.1/P896.2 Futurebus Working Group and the IEEE P1496/P1156 Rugged Bus Working Group.

A motion to merge all Futurebus and Rugged Bus activities under a single organizational structure was unanimously approved during a joint Futurebus/Rugged Bus Workshop in San Diego November 28 through December 1.

The WGs will cooperate to produce a single set of specifications to meet the combined commercial, industrial, and military requirements for the next generation backplane bus architecture for multiprocessor systems based on a platform of a revised and extended IEEE 896.1-1987 Standard.

Planned extensions to the standard include wider address/data paths, a fast pipelined transfer mode, and provisions to support real-time, maintainable, and high availability systems.

A computer bus is the electronic interconnection that enables such com-

puting elements as processors, memories, and I/O to communicate within a computer.

"This bus standard will be as significant to the world of computer hardware as the birth of Unix was to computer software," said Paul Cook, chair of the Rugged Bus WG.

"The advantage of having one international computer bus standard is that industry will be able to create hardware modules having a common hardware interface, just as Unix allows software modules to be created for a common software interface," Cook said.

"Futurebus is unique among buses because it is designed explicitly to be extensible," said Paul Borrill, chair of the Futurebus WG. "Hooks are provided for extensions needed by future technologies."

This standards activity has been endorsed by two influential organizations, the VMEbus International Trade Association (VITA) and the US Navy Next Generation Computer Resources (NGCR) program. Each has announced its intent to use the standards for its next generation of products and systems.

"Use of this new computer bus by VITA and the Navy, along with the

technical support these organizations provide, will insure its position as the dominant standard bus in the mid through late 1990s and beyond," Cook said.

VME (IEEE 1014) is the most successful 32-bit industry standard bus in use today. Computers based on this standard are used in every imaginable embedded and data processing application, including robotics, process control, factory automation, workstations, telecommunications, office automation, guidance and control, automatic test equipment, computer aided manufacturing, and numerous others.

NGCR is a program within the Space and Naval Warfare Systems Command that is Congressionally mandated to make maximum use of commercial standards and state-of-the-practice technology on all future Navy systems. The Naval Air, Sea, and Supply Systems Commands concurred with the Futurebus Standard selection.

The merger will be effective upon approval of the sponsoring organizations, the Computer Society's Technical Committee on Microprocessors and Microcomputers, the Microprocessor Standards Committee, and the IEEE Standards Board.

Lack of standards threatens information age, says Bellcore executive

Movement toward worldwide information networking and the vast capabilities it will provide are being hampered by a lack of agreement on standards that permit equipment and software to interact, according Irwin Dorros of Bellcore.

"Without universal interworking standards, there is no information age," the executive vice president told a Tel Aviv audience at the Ninth International Conference on Computer Communication sponsored by the IEEE Israel Section. "Interface standards are essential to control what could be chaos in a world of many suppliers, many users, many functions, and many tech-

nologies."

Dorros urged conference attendees to strive toward a systems approach in software and hardware architectures. He said virtually every major business and institution already has functioning information networks tailored to their unique strategic needs.

"We have reached a stage where islands of information networks exist," said Dorros, adding that the proliferation of entities dedicated to dealing with network interworking demonstrates the need for universal interworking standards.

The speaker said the International Standards Organization (ISO) model

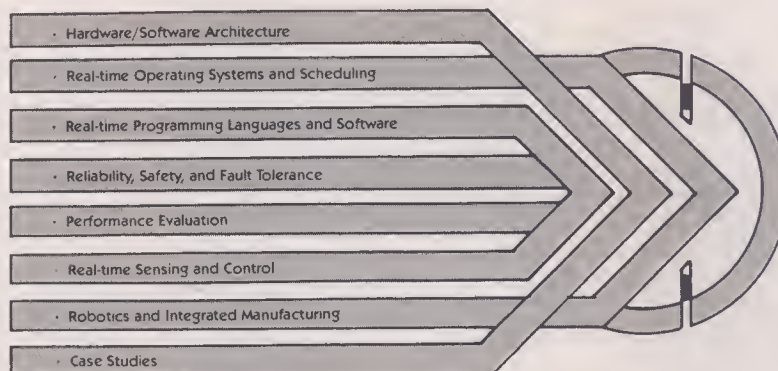
and the Open Systems Interconnection (OSI) concept are critical to guiding software designers and suppliers.

To speed the arrival of a truly universal information age, Dorros called for a user-driven international systems approach to hardware and software network standards instead of the "painfully slow" step-by-step open standards process.

He said information is growing at an exponential rate and the key challenge will be determining how to organize "incredible masses" of data, maintain their integrity, provide appropriate security, and manage directed rapid retrieval.

CALL FOR PAPERS

10th Real-Time Systems Symposium



December 5-7, 1989
Los Angeles, California

Sponsored by the IEEE Computer Society Technical Committee on Real-Time Computing.

The Symposium will explore computer systems used in real-time applications. Of particular interest are papers, especially case studies, on the control of parallel and distributed processors, sensors, and output devices to achieve real-time performance.

Both formal paper and panel discussion sessions are planned. Papers and panels are solicited in areas related to the theory, design, implementation, and evaluation of real-time systems including, but not limited to the topics above.

SYMPOSIUM COMMITTEE

General Chairman

Prof. Al Mok
Dept. of Computer Science
TAY 3-140C
University of Texas at Austin
Austin, TX 78712
(512) 471-9542
ArpaNET: mok@cs.utexas.edu

Treasurer

Dr. Walter Heimcrdinger

Program Chairman

Dr. Douglass Locke
IBM Systems Integration Division
MD 0212
Bodley Hill Rd.
Owego, NY 13827
(607) 751-4291
ArpaNET: cdl@cs.cmu.edu

Local Arrangements Chairman

Dr. Earl Swartzlander

Program Committee

Prof. Susan Davidson
Dr. Marc Donner
Dr. Flaviu Cristian
Prof. Joseph Leung
Prof. Jane Liu
Prof. Trevor Mudge
Prof. Krithi Ramamritham
Dr. Lui Sha
Prof. Kang Shin
Prof. Jay Strosnider
Prof. Horst Wedde
Prof. Wei Zhao

PAPER SUBMISSION REQUIREMENTS

SYMPOSIUM TIMETABLE:

Four Copies of Manuscript
Acceptance Letters Sent
Camera-Ready Papers (IEEE mats)
Symposium

May 1, 1989
July 15, 1989
September 1, 1989
December 5-7, 1989

Outstanding papers will be considered for publication in the journals of the IEEE Computer Society. For more information, contact the General Chairman



IEEE COMPUTER SOCIETY



THE INSTITUTE OF ELECTRICAL
AND ELECTRONICS ENGINEERS, INC.

Computer Society passes membership milestone

The Computer Society recorded its 100,000th member sometime late in December 1988, according to figures released by the IEEE at press time.

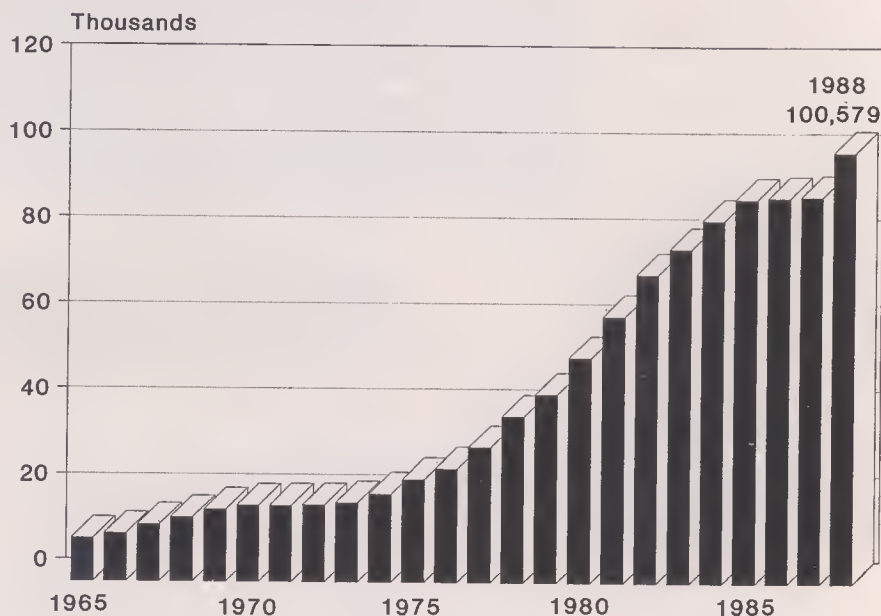
Barry Johnson, who is with the University of Virginia and serves as the society's vice president for membership and information, made the announcement. Johnson credited IBM's Merlin Smith, his predecessor in the society post, with leading the effort to achieve the 100,000-member milestone.

"This is truly a testament to the energy, dedication, and teamwork of a lot of people over the last two years," he said.

Johnson went on to acknowledge the efforts of the Membership Committee, chaired by Ron Williams of the University of Virginia, and the staff of the society's Membership and Circulation Promotion Department, led by Christina Champion.

"Their creativity and hard work were crucial," observed Johnson.

As shown in the accompanying figure, the society reached 100,579 members by year-end 1988, following a three-year period of essentially flat growth. Johnson, who himself served as Membership Committee chair under Smith in 1987, attributed the leveling-off period partly to weakness in the



IEEE Computer Society membership, 1965 to 1988.

computer industry and partly to reduced membership promotion budgets in 1985 and 1986—both tight budget years.

By comparison, the IEEE reached

304,351 at year end, and the Communication Society, the next largest IEEE society, reached 28,787. The Association for Computing Machinery reported 72,851 members at year end.

Eckert-Mauchly Award nominations sought

Nominations are being sought for the 1989 Eckert-Mauchly Award, an annual joint presentation of the IEEE Computer Society and the ACM for outstanding contributions to the field of computer architecture.

Each nominator should present the case for his or her candidate in a letter. The case may include letters written by others in support of the nomination.

Nominations, as well as any questions about the award or the nomination process, should be addressed to G.J. Lipovski, Chair, Eckert-Mauchly Award Committee, c/o Code 52Li, Naval Postgraduate School, Monterey, CA 93943-5000, phone (408) 646-2095.

Prior Eckert-Mauchly Award winners include Maurice Wilkes, C. Gordon Bell, Gene Amdahl, John Cocke, and

Daniel Siewiorek.

Applications received through April 1, 1989, will be considered for the 1989 award, and applications received after that date will be considered for later awards, Lipovski said.

The 1989 award will be presented during the 16th International Symposium on Computer Architecture in Jerusalem, Israel, May 28 to June 1.

Computer Society calls for nominees, sets down 1989 election timetable

Edward A. Parrish, Jr., Nominations Committee Chair

The Nominations Committee of the IEEE Computer Society is inviting suggestions from the membership for Board of Governors nominees to serve terms starting January 1, 1990. Seven board seats will be filled during this fall's membership election, with the top vote-getters winning three-year terms.

1, as past president and chair of the Nominations Committee, must receive suggestions for Board of Governors nominees by March 17, 1988. The suggestions must be accompanied by biographical information and should include facts about the nominee's past or present participation in the society's activities. These materials should be sent to Edward A. Parrish, Jr., Vanderbilt University, Box 1826 Sta. B, School of Engineering, Nashville, TN 37235.

Also in this year's election, the membership will select first and second vice presidents, who will each serve in 1990, as well as the individual who will be the president-elect in 1990, president in 1991, and past president in 1992.

Election timetable. The following timetable for nominations and elections (for events following the committee's March 17, 1989, nominations deadline) was approved by the Board of Governors at its November 18, 1988, meeting:

April 21—Nominations Committee sends its report to the Board of Governors.

May 9—Final date for receipt of petitions by the society's secretary, Michael Evangelist, signed by members of the 1989 Board of Governors for additional nominees for the board and for officers.

May 19—Candidates for the board and officers selected at Board of Governors meeting in Pittsburgh.

July—*Computer* lists the slate of candidates selected by the board and calls for additional petition candidates from the membership. (The process allowing members to nominate additional candidates by petition is described below.)

July 7—Position statements, biographies, and photos of the candidates selected by the Board of Governors are due at the society's Los Alamitos office.

July 31—Petitions with candidates' position statements, biographies, and photos are due at the office of the society's secretary, Michael Evangelist, MCC, Kaleido II Bldg., 9390 Research Blvd., Austin, TX 78759.

September—*Computer* carries the position statements, biographies, and photos of the candidates.

September 1—Ballots are mailed to the membership.

October 13—Last day for receipt of marked ballots.

December—*Computer* publishes the election results.

Petitioning nominations. Petitions to add nominees to the list of candidates previously selected by the Board of Governors for the board or for officers must:

- Set forth the office, the starting date of the office, and the name of the candidate.
- Contain, for board nominees, the signatures of at least 50 voting members of the society, with each member eligible to sign only one Board of Governors' petition and,

for officer nominees, the signatures of at least 250 voting members of the society, with each member eligible to sign one petition for each office. Each signature must be accompanied by the printed name of the signer. To avoid ambiguous identification of each signer, it is recommended (although not required) that the signer's membership number accompany his or her signature.

- Be accompanied by a statement signed by the nominee that he or she is willing and available to serve, if elected.
- Be received by Secretary Evangelist on or before July 31, 1989.

To qualify as a candidate for the board, a nominee must be a member of the society, must agree to seek significant involvement in society activities such as chairing a committee or serving as editor-in-chief or society representative, etc., and must meet other requirements as stated in the constitution and bylaws.

To qualify as a candidate for an officer position, a nominee must hold a member grade or higher in the IEEE, must have been a Computer Society member for the preceding three years, and must meet other requirements as stated in the constitution and bylaws.

Petitions can also be submitted to the secretary by Compmail + . To be counted as a signature on a Compmail + petition, each person signing the petition by Compmail + must originate a message stating his/her support of the nomination of the individual concerned and transmit it to Evangelist. His Compmail + ID is m.evangelist.

Simulation TC publishes joint newsletter with ACM's SIGSim

Sallie Sheppard, Texas A&M University

Simulation Digest has launched publication as the new joint newsletter of the IEEE Computer Society Technical Committee on Simulation and the ACM Special Interest Group on Simulation. It replaces *Modeling*, the old TC newsletter, and SIGSim's *Simuletter*.

Simulation Digest, which appears quarterly, includes officers' reports and announcements from both groups, plus correspondence and short contributed papers.

The newsletter is one of several projects the two organizations are pursuing jointly. Because of similarities in purpose, TC and SIG members have found a lot of common ground for cooperation. For instance, the society and ACM cosponsor several conferences, including the Winter Simulation Conference and the annual Simulation Symposium. In fact, one of the 1989 issues of *Simulation Digest* will include the annual symposium's proceedings.

At the annual TC joint meeting with SIGSim during the Winter Simulation Conference, representatives discussed plans for a new conference on simulation in manufacturing.

Simulation is very interdisciplinary in nature, making it an ideal area for joint Computer Society/ACM activities. TC Chair Paul Fishwick and SIGSim Chair Stephen D. Roberts concurred that joint activity has greatly strengthened the ability and capacity of both groups

to collect and distribute simulation materials to members and also to apportion the burden of producing such materials.

Fishwick also announced the existence of an electronic news group, like-

wise called Simulation Digest. It is available as a forum for use by researchers. This forum provides real-time interaction among colleagues all over the world.

Directions on how to access this net-

work are available by circling reader service No. 185 or contacting Paul Fishwick, Department of Computer and Information Science, University of Florida, Gainesville, FL 32611, phone (904) 335-8036.

NEWS FROM THE COMMITTEE ON PUBLIC POLICY

Position paper on computer viruses planned

Ralph J. Preiss, COPP Chair

During its November 18 meeting in Florida, the IEEE Computer Society Board of Governors directed the drafting of a society position paper on computer viruses for review at the board's next meeting, March 3 in San Francisco. It falls to the Committee on Public Policy to create and polish a draft and to see it through to completion.

We plan to first present a draft (after a few iterations) to the COPP Steering Committee at its February 28 meeting. COPP will then pass it on to the Membership and Information Board, which will comment on it and present a revised draft to the Board of Governors for consideration.

Since no one so far has volunteered to serve as chair for the Computer Ethics Subcommittee, I, as COPP chair, will meanwhile coordinate development of the draft. Please send your ideas and comments to Ralph Preiss, 12 Colburn Drive, Poughkeepsie, NY 12603.

In addition to the ideas presented by 1988 Computer Society President Edward Parrish, Jr., in the Los Angeles Times (see p. 98 of the January 1989 issue of *Computer*), I have gleaned the information that follows from notes received from COPP correspondents, especially Joshua Stern, Robert Moeller, Oscar Garcia, and Joe Urban. Somehow, the pieces, together with your inputs, must represent the consensus of the membership of this society when we publish our position statement.

The following is a preliminary list of the kinds of points that should be brought out in the position paper:

(1) When we joined the Computer Society, we automatically pledged to adhere to the IEEE code of ethics, which among other items, respects the property rights and privacy of others.

(2) A computer system and the data it controls are considered the property of the owner or the entity that pays for its use.

(3) The owning entity or its agents provides access rights to others to add or remove data, programs, or hardware to the computer system in the normal operation of the entity's functions.

(4) Any unauthorized entry into the system or the data controlled by the system should be considered an invasion of privacy, as if a lock had been broken or an unlocked door had been entered by an unauthorized person.

(5) Even the unauthorized reading of data controlled by a computer system should be considered an invasion of privacy, similar to tampering with mail or wiretapping.

(6) The Computer Society's role, in representing the technical interest of the society at large in which we live and representing our members' interests in promoting projects employing them, takes the problem of computer viruses very seriously.

(7) *Computer virus* is the name given to a computer program which, when activated, deliberately interferes with normal computer operations and can have the effect of damaging computer systems and the data they manipulate.

(8) Since computers are now tied together through communication lines into networks connecting businesses with homes and research institutions, a computer virus can spread from computer to computer through its own ingenuity or because the program to which it is attached is transmitted for use by others under normal operation.

(9) It takes highly trained professional engineers and programmers to build today's computer systems and networks and to keep them operational.

(10) Computer Society members are often involved in the design of the critical functions a computer virus needs to attack to sabotage a system's operation.

(11) The same knowledge and position required to build a smoothly functioning computing system is needed to create a computer virus.

(12) It is in the best interest of our members to make computer systems as secure, private, and reliable as possible,

because their own lives and fortunes are at stake.

(13) As part of testing operations, it is legitimate to build such viruses in controlled experiments either to protect against invasion by another virus or to measure and minimize the effects of possible damage in case an invasion does take place.

(14) Deliberately planting a virus into an unsuspecting system should be considered an unethical practice.

(15) The sabotaging of a computer system by a computer virus can take many forms. It can cause program errors by tampering with programs. It can cause errors in data by tampering with files. It can tie up computer resources by filling working memory or archival files with scratch data. The scratch data or copies of the virus can also spill over into the computer network attached to the system. (Some like to refer to these categories of failures as caused by Type I, Type II, etc., viruses, while others refer to them simply as bugs, worms, viruses, etc.)

(16) All the additional computing performed by a virus activation rob the system owner of valuable computer time, thus preventing the entity from performing its normal functions.

(17) In a position paper, we must address the technical issue of computer viruses as well as the human issue of the ethics involved in the unauthorized intrusion and possible damages caused by the application of computer viruses to computer systems and networks.

(18) We must not get into the controversy involving who are "good hackers" and "bad hackers," but instead discuss how it is unethical and even criminal that some individuals with high technical skills plant viruses into unsuspecting systems simply as an ego trip.

(19) We might recommend some kind of exposure or punishment for the culprit, such as expulsion from the society or voiding passports or Social Security numbers for life.

Please let me have your responses and suggestions.

UPDATE

Contributions to Update are welcome. Send news of industrial or university research and of public policy or professional issues to Update Editor, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720, or to Bruce Shriver, Editor-in-Chief, Dept. of Decision Sciences, University of Hawaii, 2404 Maile Way, Honolulu, HI 96822.

Changes urged in US science and technology budget process

The current federal budget process does not allow Congress and the executive branch to fund vital national science and technology programs at a time of federal deficits, a special committee of the National Academy of Sciences, National Academy of Engineering, and Institute of Medicine told the US House and Senate Budget Committees December 20, 1988.

The process "has not shortchanged science and technology nor has it failed to make choices when necessary," the committee said, but the nation must adopt a "cross-agency perspective" to "ensure that S&T budgets are more reflective of demonstrated needs and national priorities." Congress allocated more than \$60 billion in science and technology funds in the most recent budget cycle.

Moreover, the committee said, science and technology programs should be leading candidates for experimentation with two-year, rather than annual, budget cycles. It also recommended a review of how Department of Defense research and development funds are reflected in overall budget statistics; "lumping together as R&D the defense and civilian agency activities now classified as such may lead to overstating the national S&T effort overall and in certain research fields," the committee wrote.

The 13-member committee comprised members of the governing councils of the three organizations, including the three presidents. The study was requested in the Conference Report on the Concurrent Resolution on the Budget for FY 1989. Congress asked the academies and the IOM for "advice on developing an appropriate institutional framework and information base for conducting cross-program development and review of the nation's research and development programs."

Currently, the committee explained, the budget process in the executive branch focuses on how S&T activities meet the goals of individual agencies, with little examination of programs that span multiple agencies. In Congress, "agency-focused" authorization and

appropriations committees review the budgets. The committee wrote that a systematic effort is now necessary to examine cross-cutting budget items apart from their impact on individual agency missions.

The committee recommended that the President, with the assistance of the science and technology advisor and the director of the Office of Management and Budget, should develop priorities in the cross-cutting S&T areas that become important factors in the initial budget instructions to federal agencies and criteria for OMB review of agency proposals.

"These additional procedures would not alter the traditional prerogatives" of department and agency heads, nor would they "detract from the role of OMB in managing the preparation of the President's budget," the committee maintained.

The final budget submission to Congress should also include a section on S&T with a summary of agency budgets and a presentation of the cross-cutting S&T priorities that guided budget preparation, the committee stressed.

The Budget Committees of the House and Senate — either as a whole or through a special task force — should examine the S&T budget submissions before those parts of the President's budget dealing with individual agency missions are sent to committees of jurisdiction, it said. The budget committees should review cross-cutting S&T activities and recommend agency budget allocations, the committee stated.

The committee proposed four categories as a framework for evaluating S&T budgets:

- (1) S&T supporting individual agency missions;
- (2) S&T supporting the nation's science and technology base, including education of scientists and engineers, basic research, and equipment and facilities procurement;
- (3) S&T projects relevant to prominent national objectives, such as industrial development and international competitiveness, AIDS,

and global environmental change; and

- (4) major new S&T initiatives such as the manned space station, mapping and sequencing the human genome, and construction of a superconducting supercollider.

For S&T projects involving only a single agency, the committee said that the process "works reasonably well and we do not propose to change it."

Projects supporting the S&T base, funded by multiple agencies, are "the bedrock of the nation's ability to use science and technology in the national interest," the committee said. They require "continual replenishment," it noted; each aspect of the S&T base should be reviewed periodically and a few areas should be singled out for consideration in every budget year. The review should be carried out by discipline or broad science area, the committee said.

S&T projects applied to national objectives should be reviewed according to whether they contribute to specific social, economic, or other objectives according to the President's priorities; address the unresolved scientific questions related to those objectives; achieve cross-agency coordination; provide both near- and long-term results; and complement state and nongovernmental programs for the same objectives, the committee suggested.

The fourth category, major S&T initiatives, will fall into one or more of the other three categories and should be evaluated accordingly, the committee concluded. However, their size and scope require "special attention" as to their merit and their impact on funding for other activities in these categories and on future S&T outlays.

The academies and the IOM are charged with advising the federal government on science and technology issues under the 1863 congressional charter of the NAS. The report, *Federal Science and Technology Budget Priorities: New Perspectives and Procedures*, is available from the Office of Government Affairs, 2101 Constitution Ave., Washington, DC 20418.

NSF announces 11 Science and Technology Centers

Eleven university-based Science and Technology Centers will receive approximately \$24.7 million during the first year of five-year awards announced recently by the National Science Foundation.

Recommended awards for the first year range from \$900,000 to more than \$4 million. NSF support after the first year depends on available funds and on satisfactory progress by each center.

The National Science Board, the policy-making body of NSF, chose the 11 centers after about 1,000 reviewers from academia, industry, NSF, and other government agencies and laboratories reviewed 323 proposals.

Applicants were required to shape their proposals around a unifying intellectual theme, to include a strong educational component, to incorporate mechanisms for stimulating the transfer of basic research knowledge to those interested in building upon it, and to establish linkages with government, industry, states, or other institutions.

NSF will monitor progress at the centers, and a comprehensive review

after the third year will determine whether centers will be funded for an additional five years. Ultimately, NSF could support successful centers for 11 years.

The centers and their proposed funding amounts are:

- The Center for Research on Parallel Computation at Rice University, \$4.1 million.
- The Center for Discrete Mathematics and Theoretical Computer Science at Rutgers University, \$1.825 million.
- The Center for High-Temperature Superconductivity at the University of Illinois, Champaign-Urbana, \$4.25 million.
- The Center for Quantized Electronic Structures at University of California at Santa Barbara, \$2.1 million.
- The Center for Particle Astrophysics at the University of California at Berkeley, \$1.825 million.
- The Center for the Development of an Integrated Protein and Nucleic

Acid Biotechnology at the California Institute of Technology, \$3.05 million.

- The Center in Microbial Ecology at Michigan State University, \$1.1 million.
- The Center for Advanced Cement-Based Materials at Northwestern University, \$1.75 million.
- The Center for Analysis and Prediction of Storms at University of Oklahoma, \$900,000.
- The Center for Photoinduced Charge Transfer at University of Rochester, \$1.65 million.
- The Center for High-Performance Polymeric Adhesives and Composites at Virginia Polytechnic Institute and State University, \$2.124 million.

NSF established the Science and Technology Centers Research Program to promote basic research that can most effectively be accomplished through centers — large-scale, complex research problems of long duration that require special facilities or collaborative relationships.

UNIVERSITY OF NORTH CAROLINA AT CHARLOTTE CHAIRPERSON DEPARTMENT OF COMPUTER SCIENCE

Applications and nominations are invited for the position of Chairperson of the Department of Computer Science. Applicants must have a Ph.D. or equivalent in Computer Science or a related field and must show a successful record of research in computer science, computer engineering, or information science. In addition, the individual must have a strong interest in teaching and research at both undergraduate and graduate level, and exhibit academic leadership qualities. The position will be at the rank of Professor with a highly competitive salary. Anticipated starting date is July 1, 1989.

The University of North Carolina at Charlotte is one of the largest institutions of the UNC System. It has 13,000 students including 1,900 graduate students in the six colleges of Arts & Sciences, Architecture, Business Administration, Education & Allied Professions, Engineering, and Nursing.

The Department of Computer Science has 24 faculty and is the largest of the five departments within the College of Engineering. It offers a B.A., a B.S., and an M.S. degree in Computer Science and over the next few years will continue the development of its research and graduate programs including doctoral level work. The university is firmly committed to providing personnel and facilities for this department including participation in a new 75,000 sq. ft. applied research center. Our immediate proximity to the University Research Park with tenants such as IBM, Kodak/Verbatim, AT&T, Bell South, etc. and our participation in the Microelectronics Center of North Carolina and the Engineering Research Center greatly enhance our education and research activities. Current faculty strengths are in the areas of artificial intelligence, computer engineering, database systems and theoretical computer science, and computer networks.

With a metropolitan population of over one million, Charlotte is the largest city in the Carolinas. Located within several hours drive of the mountains and the ocean, Charlotte has a moderate climate, attractive neighborhoods, quality schools, and a multitude of cultural and recreational opportunities. Charlotte Douglas International Airport is the third busiest airport in the South.

Nominations and letters of application including a resume and names of four references should be addressed to: Dr. George Rent, Chairperson, CSCI Search Committee, College of Engineering, UNC Charlotte, Charlotte, NC 28223. Initial screening will begin February 1, 1989, although applications will be accepted until the position is filled.

UNC Charlotte is an equal opportunity/affirmative action employer.

University of Nebraska-Lincoln Center for Communication and Information Science Director

Seek director for Center for Communication and Information Science. The position is available July 1, 1989. Recently established with State Research Initiative funds, the Center has been created to strengthen the University's research program in this area and to be catalytic to economic diversification and development in Nebraska.

Must possess Doctorate or Ph.D. and be qualified for appointment as a tenured faculty member in the Departments of Computer Science and Engineering, Electrical Engineering, or Mathematics and Statistics. Should have a strong record of research accomplishment and a demonstrated ability to manage interdisciplinary organized research activities. Rank and salary will be commensurate with qualifications.

UNL is a land-grant, comprehensive research university and the flagship institution in the University of Nebraska system. It has a wide variety of computing resources linked by a sophisticated campus-wide network. UNL is the lead institution in the NSF-funded regional network MIDnet, and a node on the NSFnet backbone.

A letter of application with resume and names of three references should be sent by March 31 (or until a suitable candidate is found thereafter) to:

S. R. Liberty, Dean
College of Engineering and Technology
W181 Nebraska Hall
University of Nebraska-Lincoln
Lincoln, Nebraska 68588-0501

Affirmative Action/Equal Opportunity Employer

Third Annual Parallel Processing Symposium

CO-SPONSORED BY THE IEEE ORANGE COUNTY COMPUTER GROUP
CALIFORNIA STATE UNIVERSITY FULLERTON

AMERICAN DEFENSE PREPAREDNESS ASSOCIATION

UNIVERSITY CENTER
CALIF STATE UNIVERSITY
FULLERTON, CA 92634

MARCH 29-31, 1989
WEDNESDAY-THURSDAY-FRIDAY
8:30AM-5:30PM

WEDNESDAY HARDWARE

Keynote: Morning

MR. WORKS, SAIC
NEURAL NETWORKS

Afternoon Sessions

Parallel Processing
Hardware

•
Topology
Architecture

•
Optical
Neurocomputers

•
Electronic
Neurocomputers

•
Semiconductor Technology
in
Parallel Processing

THURSDAY SYSTEMS SOFTWARE

Keynote: Morning

Dr. Quinn
Univ. of New Hampshire
Parallel Algorithm

Afternoon Sessions

Parallel Processing Algorithms

•
Correctness, Debugging in
Parallel Software

•
Parallel Operating Systems

•
Performance Evaluation

•
Neural Networks for Pattern
Classification

•
Neural Networks for Con-
strained Optimization

•
Network Performance
Enhancement Algorithm

FRIDAY APPLICATIONS

Keynote: Morning
Dr. Lewis Tucker

Thinking Machines Corp.
Connection Machine
A Massively Parallel Machine

Afternoon Sessions

Optical Digital Parallel Processing

•
Local/Distributed Network
Applications

•
Parallel Processing Applications

•
Military Neural Networks
Applications

•
Commercial Neural Network
Applications

•
Far Out Topics
(Hardware and Software)

•
Data Base-Knowledge Base

Early Registration: Before Feb. 28

General Fees

Computer Society/IEEE/ACM/ADPA \$125 or \$50/day.

Non-Members \$175 all or \$65/day.

Full Time Students \$15 all or \$5/day

Name _____

Address _____

Home Telephone _____

Affiliate: IEEE Computer/IEEE/ACM/APDA: Member # _____

Fee Enclosed: March 29 March 30 March 31

Make checks payable to: OC IEEE Parallel Processing Symposium

Send to: Dr. Larry Canter,
Computer System Approach, Inc.
1140 S. raymond Ave, Suite B
Fullerton, Ca 92631

Registration After Feb. 28

General Fees

Computer Society/IEEE/ACM/ADPA \$150 or \$75/day.

Non-Members \$200 all or \$75/day.

Full Time Students \$25 all or \$10/day

Company _____

Company Tel. # _____

General Chairman:
Dr. Larry Canter, CSA
Tel: 714 738-3414

Co-Chairpersons:
Dr. John Clymer, CSUF
Dr. S. Ghazenshahi, CSUF

Technical Co-Chairman:
Dr. Tilt Thompkins, Northrop
Dr. Harold M. Stoll, Northrop

Chairman Referred Papers:
Dr. Howard Jelinek, EDA
714 751-5992



IEEE COMPUTER SOCIETY



THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.

CALL FOR PARTICIPATION

SECOND WORKSHOP ON WORKSTATION OPERATING SYSTEMS

Sponsored by:



IEEE Computer Society - Technical Committee on Operating Systems

Asilomar Conference Center, Pacific Grove, CA.

September 27-29, 1989

The Second Workshop on Workstation Operating Systems (WWOS-II) will bring together a number of researchers and developers to discuss their work and experiences in workstation operating systems. As workstation technology and man-machine interfaces advance, the operating system must evolve. Potential topic areas include:

- New workstation operating systems
- OS changes for the workstation environment (e.g. graphical interfaces)
- Scheduling and resource management
- Workstation architectures
- Multi-processor considerations

To facilitate the dialog that is so valuable in a workshop setting, attendance will be limited to 60 active workers in the field.

Registration requests should be addressed to Joseph Boykin, the General Chair. This request should include a 1-3 page position paper describing the participant's interest and future directions related to workstations

and workstation operating systems as well as insights and lessons gained from recent research and practical experience.

Those interested in presenting the results of their research, or current work-in-progress, should submit a 3-5 page paper to Luis-Felipe Cabrera. Papers must be submitted by April 15, 1989. Acceptance will be made by June 1 and final copy is due by July 1, 1989. Selection of papers will be made by the program committee:

David Anderson, *UC Berkeley*

Kenneth Birman, *Cornell University*

Anita Borg, *DEC WRL*

Luis-Felipe Cabrera, *IBM (chair)*

Sam Chanson, *U. of British Columbia*

Robert Hagmann, *Xerox Parc*

Paul Leach, *Apollo Computers*

Andy Hisgen, *DEC SRC*

Sape Mullender, *CWI, Amsterdam*

Ike Nassi, *Encore Computer*

Mike Powell, *SUN Microsystems*

David Presotto, *AT&T Bell Laboratories*

Bud Tribble, *Next Computers*

John Wilkes, *Hewlett Packard*

Registration, including a copy of the proceedings:

Computer Society Member

Non-member

Student Member

Pre-registration

\$160

\$210

\$95

On-site

\$215

\$275

\$125

Lodging:

\$55 per night for double occupancy room, breakfast, lunch, and dinner.

General Chairman: Joseph Boykin
Encore Computer Corp., 257 Cedar Hill Street
Marlborough, MA 01752
(508) 460-0500 x2720
ARPA: boykin@encore.com
UUCP: encore!boykin

Program Chairman: Luis Felipe Cabrera
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120
(408) 927-1838
ARPA: cabrera@ibm.com
UUCP: ucbox!cabrera

PRODUCT REVIEWS

Editor: Richard Eckhouse, MOCO, Inc., PO Box A, 91 Surfside Rd., Scituate, MA 02055; Comppmail+, r.eckhouse

A Turbo family portrait

Noah Davids, Stratus Computer

This review looks at and compares three of Borland's Turbo languages: Basic, Pascal, and C. It concentrates on language and user interface features as well as installation and customization considerations. I have included some compilation and execution statistics, but I made no attempt to create a complete suite of benchmarks.

Turbo Basic, Turbo Pascal, and Turbo C have a marked family resemblance, but on the order of first cousins or, in the case of Turbo Basic, second cousins, not siblings. I also noticed a difference in maturity, with Turbo C being the most mature, that is, more sophisticated in allowing you to define your environment. Next in age comes Turbo Pascal, followed by Turbo Basic.

A look at the licence agreement and support mechanism shows that these three languages share at least one grandparent with royal blood. The person who wrote the licence agreement is at least a prince. It allows you to place the language and associated programs and data files on any number of computer systems as long as there is no possibility that they can be run simultaneously on two or more of those systems.

Borland provides telephone support from 9 to 5 Pacific time. Unfortunately, it's not an 800 number. I found the best time to call was early in the morning. At that time I was never put on hold. I found all the support people knowledgeable.

Borland also provides a set of forums on CompuServe and BIX for the distribution of information and questions and answers. The packages come with an order form to get a free CompuServe introductory pack and a \$15 credit. Unfortunately, I do not have access to either of those systems, so I cannot comment on their content or usefulness.

All three languages are for the IBM PC, XT, AT, or compatibles and require MS- or PC-DOS 2.0 or later. Turbo Basic requires 320 Kbytes of memory, while

Turbo Pascal and C require 448 Kbytes of memory. Turbo Basic sells for \$99.95, while Turbo Pascal and Turbo C both list for \$149.95.

Turbo Basic 1.1

The Turbo Basic 1.1 package consists of a 466-page manual and two 5.25-inch disks. The manual gives a brief description of Basic's beginnings and the differences between an interpreter and a compiler. It claims that it is not a DOS or Basic tutorial, and it lives up to that claim. It is, however, an excellent reference manual for the language.

Installation is a breeze. Just copy the files off the two floppies. All the sample programs compiled and ran without error and provided some useful examples for how to do things.

Copying all 56 files from the two disks took up 671,744 bytes. A minimum configuration — which consists of only the Turbo Basic exe file (tb.exe) and a configuration file (tbconfig.tb) — requires 213,730 bytes. The help file (tbhelp.tbh) adds another 41,029 bytes.

Turbo Basic comes with a customization program that allows you to customize several different things, the most important (for me) being use of colors and keystrokes for the editor commands. Unfortunately, the configuration program senses the type of display card and comes up in color. This presented me with an unreadable display because I have a CGA display card and a monochrome monitor. However, using the screen images in the manual I ran through the menus and selected the black and white option. If you have a color monitor, you can use the setup option on the menu bar from within Turbo Basic's integrated environment to set any combination of colors you choose.

Between home and work I deal with several different editors and suffer from

crossed editor syndrome — right command, wrong editor. I was looking forward to Turbo Basic allowing me to customize the editor so I could eliminate or at least reduce the problem. Alas, it was not to be. The editor has two sets of commands, a primary and secondary. You can change only the secondary set with Turbo Basic, and your changes cannot conflict with anything in the primary set. This eliminated most of the changes I wanted to make. In addition, your changes cannot conflict with the set of global hot keys. I ended up leaving the commands unchanged.

The Turbo Basic integrated environment contains the editor, compiler/linker, and a host of menus for setting switches and options. It has five windows, one containing the menu bar. You select options from the menu bar and its submenus by pressing the first letter of the option name or, in some instances, a global hot key. These windows can be moved, removed, resized, and overlapped.

It would have been nice to have Turbo Basic recognize and use a mouse, but I found the environment easy and enjoyable to work in. A context-sensitive help facility does a good job until it gets into the editor. In the editor, all the help concerns the editor, with nothing about the syntax of any of the language statements.

The Turbo Basic Language is significantly richer than the original Basic or the minimum Basic defined by the ANSI committee. It is also richer than the BasicA or GWBasic interpreters distributed with PC- and MS-DOS. (See the accompanying sidebars for a list of the supported data types and control flow and conditional compilation statements.) Borland took care to insure compatibility with BasicA and GWBasic. The manual contains a chapter on the few minor differences and significant extensions.

I found certain aspects of Basic

bothersome. First, statements must either occupy one line only or continue by having the last character on the line be an underscore. I kept forgetting the underscore or, worse, trying to use variable names with an embedded underscore, like `last_name`. Second, variables are not declared, so spelling a variable wrong does not produce a compilation warning. Finally, Basic uses suffixes on the variable name to indicate the data type. `A$`, `A%`, and `A` are all different variables, and I kept leaving off the suffix.

Simple syntax errors can cause the compiler to generate the most obscure error messages. All the error messages appear in the manual with explanations. Most of the time that helped; sometimes it didn't. For example, "write `B(1%, J%);`," produces the error message "Unknown identifier/Syntax error." The manual

explains this with "Something is incorrect on the line, the compiler could not determine the proper error message." The problem is the semicolon at the end of the line. Still, by the end of my experimentation with the language I had learned to avoid or recognize all the error messages, and they no longer slowed me down.

The compiler is fast, but it stops at the first error. It drops you back into the editor at the line where it thinks the error occurred. Usually you are on the correct line, but sometimes you're on the next executable line. One nice thing is that the compiler's progress is updated in the message window every 20 lines.

Programs created by Turbo Basic automatically test for the math coprocessor chip and will use it if available. Programs use emulation code for when the chip is not available. A switch on the

menu bar tells Turbo Basic to compile a program without the emulation code. To execute, such programs require the math coprocessor chip to be available.

Compilation uses the large memory model. Although you can control the segmentation of the code segments, you must accept Turbo Basic's segmentation of the data segments. One data segment holds all scalar variables, another holds all string variables, and each array gets its own segment.

Turbo Basic has the ability to use *include files*, separate files of source text included in the main program during compilation. If the include file has an error, it will appear in the editor window with the cursor at the problem point. However, after you have made the correction, you must save the file and load the main program before compiling again.

I found that Turbo Basic supplies all the tools you need to write programs for the PC. Command-line arguments given when your program is invoked are easily available, as are the contents of the environment table. Your program can invoke common DOS functions, including changing directories, deleting or renaming a file, and shelling back out to DOS. A set of `ON <interrupt>` statements allows you to easily specify interrupt handlers for such things as a com, keyboard, and timer interrupts; and the `call interrupt` statement allows you to issue your own interrupts. Turbo Basic also includes a nice set of graphic primitives as part of the language.

One thing I did find surprising was the lack of text-mode window support. There is no way to clear a region of the screen, just the entire screen. There is not even a `clear_to_end_of_line` function.

One of Turbo Basic's five windows is a trace window. With trace turned on, the name of every procedure, function, and statement label displays in the window when the procedure, function, or statement executes. This is a much cleaner approach than putting print statements in the code. It is, however, the only debugging tool provided.

In summary, the flow of control extensions in Turbo Basic have changed Basic from a language you have to program around to a language you can program with. The simplicity of the language and its environment make it easy to learn and use. The DOS functions and interrupt support will allow you to write sophisticated programs. However, I missed the ability to define record structures, and the lack of variable declarations contributed to some difficult-to-locate bugs.

Data types

In the accompanying table, which lists data types for the three languages, the column entries indicate a keyword or other syntactic element that denotes the object. A blank indicates that the object is not available in the language.

	Basic	Pascal	C
Integers			
0 .. 255		byte	unsigned char
-128 .. 127		shortint	char
0 .. 65535		word	unsigned int
-32768 .. 32767	%	integer	short
-2147483648 .. 2147483647	&	longint	long
-2E63+1 .. 2E63-1		comp	
Reals			
-8.43E-37 .. 3.37E38 (4 bytes)	!		
-1.5E-45 .. 3.4E38 (4 bytes)		single	
-3.4E-38 .. 3.4E38 (4 bytes)			float
-2.9E-39 .. 1.7E38 (6 bytes)		real	
-5.0E-324 .. 1.7E308 (8 bytes)		double	
-4.19E-307 .. 1.67E308 (8 bytes)	#		double
-1.7E-308 .. 1.7E308 (8 bytes)			
-1.9E-4951 .. 1.1E4932 (10 bytes)		extended	
-3.4E-4932 .. 1.1E4932 (10 bytes)			long double
Characters			
1		char	char
1 to 255		string	
1 to 32,767	\$		
Boolean			
		boolean	
Pointer			
		^	*
Aggregates			
Arrays	dim	array of	[]
Records		record	struct
Variable records		case of	union
Enumerated		()	enum
Set		set of	
User defined			
		type	typedef

Turbo Pascal 5.0

The Turbo Pascal 5.0 package includes a 350-page user's manual and a 493-page reference manual. Unlike the Turbo Basic manual, the Turbo Pascal reference manual includes a good tutorial on Pascal. It is also an excellent reference manual.

Executing an install program copies all the files from the Borland floppy disks onto your hard disk. You can specify the main Turbo Pascal directory as well as the directories for the graphics, documentation, and example files. The program catches you when you insert an incorrect disk. It also expands all the examples from their compacted format. The sample programs compiled and ran without error. They provided some useful examples, especially the impressive graphics samples with their excellent examples of how to use the graphics routines.

The 79 files from the three 5.25-inch disks took up 1,014,272 bytes. A minimum configuration consisting of the Turbo Pascal exe file (turbo.exe), the unit library file (turbo.tpl) and a configuration file (turbo.tp) requires 194,043 bytes. The help file (turbo.hlp) takes another 164,999 bytes. For graphics programs you will also need the graphics unit (graph.tpu) at 32,192 bytes and the driver for your video hardware; the EGA/VGA driver (egavga.bgi) is 5,363 bytes.

The Turbo Pascal customization program (tinst.exe) resembles that of Turbo Basic. However, this program has a "/B" switch to tell it to come up in black and white mode. Color customization is done in this program rather than in the integrated environment as in Turbo Basic. The editor customization is also smarter. Like Turbo Basic, there is a primary and secondary set of editor commands, and you can only change the secondary set. However, conflicts between the primary and secondary sets are allowed; the secondary set of commands has precedence. Conflicts between the global hot keys and the secondary set of commands are also allowed. Again, the secondary set of commands has precedence.

The Turbo Pascal integrated environment contains an editor, compiler/linker, a host of menus for setting options and switches, and a source language debugger. It normally shows two windows: the editor window and either the watch or the output window. The watch window starts out with one line. It grows and shrinks depending on how many variables the debugger is watching. The output window shows a portion of your output screen. Its size is configured while in the tinst.exe program. The watch window is shown by default. When you run a

program, the screen automatically switches to show the entire output screen unless you are also debugging the program. In that case, the screen switches between the output and debugging (editor/watch windows) screens.

When both the editor and watch windows show on the screen, three lines serve as borders, as do columns 1 and 80, and another three lines communicate status information. You can zoom a window so that only one window shows on the screen, which also eliminates the borders.

If you have an EGA or VGA system, you can also use their 43- or 50-line modes. Line 1 has Turbo Pascal's menu bar, and the last line indicates the functions of some of the current hot keys.

As with Turbo Basic, it would have been nice to have Turbo Pascal recognize and use a mouse, but I found the environment easy and enjoyable to work in. The context-sensitive help does a

good job outside of the editor and an excellent job in the editor. To get help on the syntax and function of any procedure or unit, just position the key over the object's name and press control-F1. You can also do the same with any language construct. If you forget the syntax of a while loop, type "while" and control-F1 and presto, there it is. To get help on the editor commands, just press the F1 key.

When the integrated environment is first entered, it will check for an EMS driver conforming to version 3.2 or later. If it finds one and also finds a free 64-Kbyte block, it will load the editor buffer into EMS memory. If you need the memory for something else, a switch will tell it not to use any EMS memory.

The manual provides a good discussion of the differences between ANSI Pascal and Turbo Pascal. I counted 49 differences, most of them ANSI restrictions not enforced. Going from Turbo Pascal to an ANSI Pascal should not cause problems

Flow of control statements

The following table lists the flow of control statements for the three languages. The column entry indicates the statement name or yes if the statement is available but has no name.

The entries for PC interrupts for Turbo Pascal and Turbo C are procedures supplied in libraries, not part of the actual language.

	Basic	Pascal	C
Conditional			
if-then-else	if	if	if
case	select	case	switch
Loop			
iterative	for	for	for
test before	while	while	while
test after	do until	repeat	do
Subprograms			
subroutines	sub	procedure	yes
functions	def fn	procedure	yes
Goto			
break	exit		break
continue			continue
goto	goto	goto	goto
multibranch	on N goto		
PC interrupts			
invoke interrupt	call interrupt	Intr MsDos SetIntVec	int86/int86x intdos/intdosx
trap interrupt	on com on error on key on pen on strig on timer	procedure interrupt	setvect interrupt ctrlbrk

Conditional compilation statements

Conditional compilation statements for Turbo Basic are

\$IF <CONSTANT>	CONSTANT is a program variable constant. If
\$ELSE	CONSTANT is nonzero, the IF will evaluate to true.
\$ENDIF	

Conditional compilation statements for Turbo Pascal are

{ \$DEFINE <NAME> }	Defines the conditional compilation variable NAME. Conditional compilation variables are either defined or undefined; they have no specific value. Conditional compilation variables are for the compiler's use; the program cannot use them. Conditional compilation variables can also be defined before compilation from one of the options on an integrated environment menu or by the use of switches from the command-line compiler.
{ \$UNDEF <NAME> }	Undefines the conditional compilation variable NAME.
{ \$IFDEF <NAME> } { \$ELSE } { \$ENDIF }	True if NAME is defined.
{ \$IFNDEF <NAME> } { \$ELSE } { \$ENDIF }	True if name is undefined.
{ \$IFOPT SWITCH } { \$ELSE } { \$ENDIF }	SWITCH is the name of a compilation switch. SWITCH is followed by a + or - to indicate if the test is for switch being on (+) or off (-). The switch+/- is the same syntax used to actually turn the switch on or off. Examples of switches are R = range checking N = use 8087 coprocessor S = stack overflow checking

Conditional compilation statements for Turbo C are

#define name value	Defines a constant or macro. The preprocessor will replace all instances of "name" with "value" wherever name occurs in the program text. Macros can be include arguments and can contain multiple statements.
#undef name	Undefines a constant or macro.
#ifdef name #else #endif	True if name is defined.
#ifndef name #else #endif	True if name is not defined.
#if condition #else #endif	True if the condition evaluates to true. Condition can be any number of defined functions, sizeof functions, and relational comparisons on defined values combined with the logical operators OR and AND.
defined name	Returns true if name is defined.
#error msg	Prints msg along with file name and line number and stops compilation.

as long as you read this section of the manual and program with portability in mind.

A few things might cause some problems in going from an existing ANSI program to Turbo. Among them are

- (1) the Get and Put functions are not defined,
- (2) there are some restrictions on New and Dispose, and
- (3) Pack and Unpack are not defined.

This portion of the manual does contain one misprint. The section's title says "Comparing Turbo Pascal 4.0 with ANSI Pascal." However, tech support has assured me that it's correct for Turbo Pascal 5.0.

Turbo Pascal has none of the language attributes that I found bothersome in Turbo Basic. I must admit, however, that I have programmed extensively in Pascal, so I might be slightly prejudiced toward the language.

Turbo Pascal includes the ability to create a separately compiled unit, linked to the main program after compilation. Units are divided into several parts. An interface section identifies all the public variables and procedures and the calling sequence to those procedures. Anything declared in the implementation section is known only to the unit.

Finally, you get an initialization section, executed before the first statement in the main program. All the non-language defined functions and procedures are implemented as units. When units are compiled, they do not use the standard MS-DOS object format. This allows fast linking of units.

Standard object files can be linked into a Turbo Pascal program, but they must follow specific rules. Turbo C has a set of switches that will cause the object file that it creates to follow the rules. The Turbo Pascal samples disk provides an example Turbo C program and compiler switch settings.

Units also help control code segmentation. Each unit must be less than 64 Kbytes in size. However, you can have as many 64-Kbyte units as will fit in your machine's memory. You can also combine a set of independent units into an overlay so that only one is in memory at any one time. The overlay manager will check for free EMS memory and, if there is enough, load the overlay file into EMS memory. This eliminates the disk I/O typical when a new overlaid unit is required.

Only one 64-Kbyte data segment exists for an entire program. If you have more than 64 Kbytes worth of data, it must be allocated dynamically on the heap. The heap can expand to take up the rest of available memory.

I found the compiler much more robust

than that of Turbo Basic. Like Turbo Basic, Turbo Pascal compiles and links a program fast. In the event of an error it will stop and position the cursor on the offending line in the editor window. It located incorrect lines accurately and always reported useful error messages. The compiler's progress appears in a large pop-up window in the center of the

screen.

Compile-time switches allow you to select between the generation of 8087/80287 code and software emulation. When using software emulation, your program will use the 8087/80287 chip if present or software to emulate it if not present. The special high-precision data types designed for the 8087/80287 also

work under emulation.

Turbo Pascal handles include files in a much friendlier way than does Turbo Basic. If your include file has an error, the program moves you to the line with the error, just like Turbo Basic. However, after correcting the error, you can recompile the main program without reloading it.

Compile and execution statistics and memory requirements

The accompanying tables present the compile and execution statistics and memory requirements for my three sample programs. These programs are not meant to be a benchmark. I used them to test various aspects of the

languages and to get a feel for the environment of each language. I kept the programs as similar as possible, using unique features in each language without changing the structure of the program.

Table 1. Mat_mult program results.

	Basic	Pascal	C
Compile/link/execute (in seconds)	4	3	27
Compile/link → exe file (in seconds)	4	4	27
Memory required (integrated; in bytes)	319,632	347,008	428,048
Memory required (exe; in bytes)	113,440	42,032	90,704
Execution time (in seconds)			
2X2			
integrated	.55	.66	.94
exe	.33	.55	.87
10X10			
integrated	8.02	11.59	17.80
exe	7.09	10.93	17.08

Table 2. Address program results.

	Basic	Pascal	C
Compile/link/execute (in seconds)	10	8	34
Compile/link → exe file (in seconds)	12	9	34
Memory required (integrated; in bytes)	331,040	360,416	419,568
Memory required (exe; in bytes)	116,496	50,592	80,304

Table 3. Sketch program results.

	Basic	Pascal	C
Compile/link/execute (in seconds)	9	8	41
Compile/link → exe file (in seconds)	11	10	41
Memory required (integrated; in bytes)	326,800	322,576	434,848
Memory required (exe; in bytes)	117,296	60,928	91,920

The Basic and C programs were compiled with the default switch settings. The Pascal switch settings, except for the heap size, also had default values. The heap size was reduced from the default of 655,360 to 10,240 in the mat_mult and address programs and 20,480 in the sketch program. I did this so enough memory would be available when executing from the integrated environment to execute the exec procedure. The exec procedure shells back to DOS. I used it so I could display a memory map and thus determine the memory requirements of the programs. Sketch required a heap larger than 10,240 to run correctly.

I ran each program from within the languages' integrated environments and by creating an exe file and running it directly from DOS. Turbo C always creates an exe file so the two times are the same. Except for the execution times of mat_mult, I obtained all times by consulting my watch.

Mat_mult is a simple matrix multiplication program. It can handle up to a 20X20 matrix of floating-point data types. It takes its input from a file, displays the two matrices in the file, then multiplies them and displays the resultant matrix. The file name is passed in as a command-line argument. The program times itself from right before it reads the data file to right after it displays the resultant matrix. See Table 1.

I do not have an 8087 chip.

The Address program implements a simple address book with a maximum of 50 names. You can display each name or search for a name based on the last or last and first names. It has a Lotus-style menu bar on the first line of the screen. This program explores a language's ability to manipulate parts of the screen as a unit and to handle the arrow keys. See Table 2.

Sketch implements a computer version of the etch-a-sketch toy. You use a mouse to draw on the screen. This program uses the graphics capability of the language and also issues software interrupts to communicate with the mouse. See Table 3.

Turbo Pascal supplies the tools you need to write programs for the PC, as well as some nice extras. Command-line arguments given when you invoke your program are easily available, as are your environment variables. Common DOS functions can be invoked by your program. Critical DOS errors are trapped and treated in the same way as noncritical errors. Never again will you get an "Abort, Retry, Ignore" message in the middle of a carefully constructed screen display.

The text window support in Turbo Pascal is excellent, and the graphics support is superb. The graphics unit includes fill and line patterns, circle, bar, a 3D bar, ellipse, pie, and rectangle functions. You also get device drivers for the most popular video boards, including EGA and VGA, and four stroked fonts.

Turbo Pascal does not have Turbo Basic's ON <interrupt> construct. Instead, it gives you the ability to create an interrupt handling routine and modify the interrupt vector to call your routine instead of the system routine. This is obviously much more flexible than Turbo Basic. It also requires a higher level of sophistication on your part.

A keep function assists you in creating terminate and stay resident programs.

Turbo Pascal 5.0 provides an excellent debugging environment. The screen moves between the output screen and the editor/watch screen based on switches that you set. By default it switches back to the edit/watch windows when a break point is hit, but you can keep it where you choose. You can set breakpoints, step through the execution of part of a program one line at a time, or set a temporary breakpoint based on the cursor position in the edit window (which you can set).

The watch window allows you to watch the values of variables as they change; the edit window shows the part of the code being executed, with the current line highlighted. You can also evaluate expressions using program variables and any built-in Turbo Pascal function or change the value of a program variable. You can display the calling stack of procedures and display the current line in any procedure on the stack.

The only function lacking that I would like to have is conditional breakpoints, which break when a variable value changes or changes to a specific value or range of values. I would not call this omission fatal or even serious, since I easily debugged my test programs, including the graphics program.

Turbo Pascal comes with several utilities. The four most useful are tpumover, grep, make, and tpc.

Tpumover moves separate unit object files into the unit library. Only one unit library is supported, but it's not necessary to have units in this library to use them.

Grep allows you to search a set of files for a particular string or regular expression. You have a fair amount of flexibility as to the results of the search. You can have just the names of the files that contain a match or the file names plus the actual lines display. The lines can be numbered or you can display lines that do not contain the string. Grep can ignore letter case and also search subdirectories. You can find uses for this tool that have nothing to do with Turbo Pascal.

Make allows you to automatically build an executable program. A make file lists the separate files that make up the program and defines which files are dependent on other files. When you "make" a program, it checks the dates of all the object, source, and include files. If it finds a source or include file modified after its object was created, it recompiles the source.

Make is another extremely useful tool. However, the syntax of the make file is complex. The manual states that creating a make file is almost like writing a program, and I would agree. Unfortunately, the program looks quite a bit like C and nothing like Turbo Pascal.

The last utility I will mention is the Turbo Pascal compiler (tpc). Along with the integrated environment you get a command-line compiler. All the switch settings available in the integrated environment are also available to this compiler via command-line option arguments. This utility allows you to bypass the integrated environment. You can create the program source with any ASCII text editor, then compile and link it with the command-line compiler. This reduces the amount of memory required to 256 Kbytes.

Turbo Pascal is a powerful system, providing all of the tools to create professional packages of varying sophistication while remaining simple enough for personal use. Its graphics support allows you to create some outstanding visual displays quickly and easily.

Reader Service 22

Turbo C 2.0

The Turbo C 2.0 package contains two manuals: a 612-page reference guide and a 425-page user's guide. Two types of packages are available, one containing 5.25-inch disks and the other, 3.50-inch disks. The package I reviewed contained three 3.50-inch disks.

Like Turbo Pascal, you install Turbo C by running an install program. This program loads the files from the disks into any set of directories that you specify. You can specify the main Turbo C directory as well as specific directories for the include files, library files, graphics library files, and examples. The program displays the names of the files as it reads them from the disks and writes them to the hard disk. This lets you know that the program is working, but doesn't tell you how far along it is.

The program is also smart enough to catch you if you insert the wrong disk and to sense the type of video board you have and come up in color if you have it. If you want to run the program in black and white mode, you can invoke it with the "/B" switch. I had no problem with the installation.

The set of demo programs disappointed me. There are only three complete demo programs. Bigdemo demonstrates all of the graphics functions. It has good comments and provides a good example on using the graphics routines. The graphics capability is the same as Turbo Pascal. The mcalc spreadsheet program uses many system functions, but includes no useful comments about how it uses them or why. The last complete demo, the standard "hello world" demo, is a one-line program that prints "hello world" on the screen.

You also get examples on how to interface with Turbo Pascal and Turbo Prolog, plus a buggy word count program that demonstrates using the debugger. The graphics and debugger examples are excellent, but compared with Turbo Basic and Turbo Pascal, Turbo C is a demo desert.

The 108 files require 2,119,885 bytes. A minimum configuration would contain all 29 include files, four libraries (to support a specific memory model), and the integrated environment exe and configuration file for a total of 500,393 bytes (assuming the small memory model). The help file adds another 222,200 bytes. For graphics programs you'll need the graphics library at 29,247 bytes and a driver for your video hardware (the EGA/VGA driver takes 5,363 bytes). Installation on a floppy-disk-based system is possible; according to the manual it requires three disks, and only one memory model will be installed.

The Turbo C configuration program resembles that of Turbo Pascal. The integrated environment also looks similar to Turbo Pascal. It has the same editor and watch/output windows and 43/50 line EGA/VGA support. It has the same excellent help system and lack of mouse support. However, it has even more options and switches that you can set

from the menu bar. One of those switches allows you to optimize the generated code for either size or speed. Another lets you generate code using the 80286 instruction set.

Turbo C supports direct manipulation of the math coprocessor chip. It provides functions for getting the status word, changing it, and setting bits in it. Like Turbo Basic and Turbo Pascal, programs created by Turbo C can automatically test for the math coprocessor chip and use it if it's available or emulate it if it's not available. By default the math library is not linked into the exe file. You have to explicitly include a function from the math library — declaring float variables is not enough. This caused one of my example programs (mat_mult) to generate an obscure, undocumented runtime error message.

Turbo C supports the ANSI standard with extensions to allow full use of the PC. You can set switches to tell the compiler to treat any extended keywords as identifiers instead of keywords. A set of checks can be made for ANSI violations and possible portability problems. A section in the manual covers extensions from the original C definition in Kernighan and Ritchie and also ANSI extensions. The ANSI extensions are all minor. You should have no trouble writing portable code and little trouble importing code written for other standard compilers.

Traditionally, C gives you plenty of rope. You can use that rope to build a strong, well designed program or to hang yourself. The choice is yours. Unfortunately, rope is a tricky building material, and sometimes you can hang yourself without meaning to. The Turbo C compiler includes checks for common and not so common errors. These checks should be helpful to someone learning C.

The ability to create separate object modules and link them together has always been a part of C environments. Turbo C is no exception. In addition, modules compiled using different memory models can be linked together. (Turbo C provides six memory models: tiny, small, medium, compact, large, and huge.)

The compiler produced correct error messages and useful warnings, as mentioned above. One giant advantage over Turbo Basic and Turbo Pascal is that it continues past the first error. By default it will continue for 25 errors and 100 warnings. You can increase or decrease those numbers as you wish. Warnings and errors appear in the message window. To locate errors in the source code, you move to an error or warning message and press return. Even after lines have been added or deleted to correct other

problems, Turbo C finds the error. It moves smoothly from file to file if errors occur in include files. If a project file defining the names of the main and submodules exists, you can recompile without reloading the main module. Like Turbo Pascal, the compiler's progress appears in a pop-up window.

Turbo C provides all the tools you need to create programs for the PC. Command-line arguments and the environment variables are readily available. Common DOS functions are available. Text window and graphics are the same as in Turbo Pascal with the exception that the Turbo C text-mode procedure does not support the 43/50 mode of EGA/VGA. The support for creating RAM-resident programs also resembles that of Turbo Pascal. Exception handling is flexible. You can write your own error handling routines for DOS hardware errors such as disk door left open, for math errors such as negative sqrt arguments, and for control break. Turbo C also has routines to print standard error messages for system errors.

A set of library procedures handles calling any DOS or BIOS interrupt. Specialized procedures call specific types of interrupts. This has the advantage of a slightly cleaner set of arguments and, of course, documentation in the Turbo C reference manual. You also get routines to disable or enable interrupts and to change the interrupt vector. Like Turbo Pascal, you use an interrupt keyword when defining a procedure.

The source language debugger looks and works the same as Turbo Pascal's source language debugger.

Turbo C comes with a host of utilities, including the same make and grep utilities as in Turbo Pascal. A librarian program allows you to create and modify libraries. Unlike Turbo Pascal, you can create and use any number of libraries. You get a command-line form of the compiler and linker and a utility for generating a new source file with include files and macros expanded out.

Turbo C is a powerful system that provides all of the tools you need to create professional packages of any level of sophistication. I think that C is harder to learn and use than Pascal and would not recommend it as a first language for someone wanting to learn to program. The greater support of the math coprocessor and 80286 chips, error processing, and memory models are the biggest functional differences between Turbo C and Turbo Pascal. The integrated compiler's ability to continue past the first error is offset by its slower

speed. The compiler's flags for common errors, ANSI violations, and portability problems will be a big plus for anyone writing portable code.

Reader Service 23

Product notes

- If you're looking for a ready reference guide and sturdy folding template for your word processor, spreadsheet, or database software, consider ThorData, Inc., 15353 N.E. 90th Street, Redmond, WA 98052-0000, (800) 423-7140. I have and use their Microsoft Word 4.0 folding template for the IBM enhanced keyboard. It's much more convenient than a manual or the flimsy template that comes with the software. The \$24.95 price tag seems reasonable for something so valuable. A dozen or so templates are available for most major software packages.

- Like most of us, I've had to recover from a loss of data on either a floppy or hard disk. Now from the creator of Mace Utilities comes *The Paul Mace Guide To Data Recovery* (Brady Books, New York), a delightful and extremely helpful working reference compendium that not only solves the data loss problems but goes on to explain the mysteries of disk systems. Users of any of the popular data recovery software (including PC Tools and Norton Utilities) will find this \$21.95 book a valuable addition to their DOS reference library.

- Looking for a way to make overhead transparencies directly from your dot matrix printer? Arkwright, Inc., Main Street, Fiskerville, RI 02823, (800) 942-5900 has a dot matrix film that is high resolution and smudge resistant, while accepting a full spectrum of colors.

- Elographics, Inc., 105 Randolph Road, Oak Ridge, TN 37830, (615) 482-4100, best known as a touch-screen manufacturer, has just introduced TouchBack. The program offers a complete set of touch applications tools so that you can modify most programs to accept touch input. A second program, TouchUp, takes snapshots of an application screen and lets the user set up touch zone boundaries. The programs work with a variety of applications such as dBase, Show Partner/FX, Basic, and batch files. The price of TouchBack is \$99.

NEW PRODUCTS

Contact or send releases to Nancy Hays, *Computer*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; Compmail+ , n.hays

Multiprocessing workstations claim Sun-4 compatibility

Solbourne Computer has announced the first models in a line of multiprocessing workstations that the company says are compatible with Sun Microsystems' Sun-4 Superworkstations. The Series4/600 Superworkstation includes one to four processors. The Series4/800 File/Compute Server contains up to four processors with up to 3.3 Gbytes of disk storage in a pair of cabinets. The processors use the Sun Sparc and will reportedly execute application programs written for the Sun-4 without porting or recompilation.

Solbourne built the multiprocessing system around its 64-bit Kbus, which according to the company operates at 128 Mbytes per second. Each processor has its own cache memory, but all share a single main memory and operating system. The processors use Fujitsu's SF9010IU Sparc 32-bit RISC CPU and

Weitek's floating-point chips.

A single-processor Series4/601 with a Series4/801 server supposedly achieves 9.5 MIPS and 1.6 Mflops of performance. A dual-processor Series4/602 with a Series4/801 hits 17 MIPS and 2.9 Mflops, while a triple-processor Series4/603 with a Series4/803 hits 24 MIPS and 3.6 Mflops. A four-processor Series4/604 with a Series4/804 server reputedly attains 30 MIPS and 4.7 Mflops.

All models come standard with 16 Mbytes of main memory, a 7-slot Kbus, a 7-slot VMEbus, an Ethernet/CheaperNet controller, two RS-423-A ports, an external SCSI mass storage interface, the SunOS operating system, Sunview, X Window System, X Window Manager, and a C language compiler.

Options include 19-inch monochrome and color monitors, a 126-key key-

board, an optical mouse, additional memory up to 80 Mbytes, up to 1.3 Gbytes of SCSI disk storage (Series4/600), up to 3.3 Gbytes of SMD storage (Series4/800), a 150-Mbyte cartridge tape drive, a Fortran 77 compiler, the GKS/C and Figaro graphics systems, and the NeWS windowing system.

A dual-processor Series4/602 with 16 Mbytes of memory, a 327-Mbyte disk, and a 150-Mbyte cartridge tape sells for \$51,400. A single-processor Series4/601 costs \$45,400. A Series4/801 server with 32 Mbytes of memory and two SMD drives sells for \$75,400.

Solbourne provides a maintenance and support plan that includes installation, training, a one-year warranty, and a year of customer support.

Reader Service 30

New VAXstations join DEC workstation family, including graphics models

Digital Equipment has added several new models to its family of VAXstation workstations. The VAXstation 3100 reputedly delivers the best price-performance ratio of any of the VAX-based workstations. The VAXstation 3510 and 3540 workstations add 2D and 3D graphics to the X Window System environment.

The VAXstation 3100 comes in two models: the desktop Model 30, configurable as a stand-alone or networked system, and the Model 40, which accommodates additional storage. It runs at 22.2 MHz and features 8 to 32 Mbytes of memory, 32 Kbytes of on-board and 1 Kbyte of on-chip cache, an SCSI port, an Ethernet controller, a 15- or 19-inch monochrome or color monitor with 1,024 × 864 pixel resolution, 105-key keyboard, and a three-button mouse.

Model 30 accommodates three 3½-inch devices or one 3½-inch device and a

95-Mbyte streaming tape drive. Model 40 accommodates three 3½-inch devices and two 5¼-inch storage devices.

The VAXstation 3100 uses the VMS and Ultrix operating systems and the DECwindows and VWS/UIS windowing systems. MS-DOS compatibility is supplied through VAXpc for VMS (which requires a separate license).

The VAXstation 3100 costs \$7,950 with 8 Mbytes of memory and a 19-inch monitor. It is scheduled for availability in March 1989.

The VAXstation 3520 contains two CVAX/CFPA processors and the 3540 has four CVAX/CFPA processors. Both have 1 Kbyte of primary, on-chip cache and 64 Kbytes of secondary cache per processor. They feature 8 to 64 Mbytes in 8 or 16-Mbyte modules; one to four 332-Mbyte SCSI 5¼-inch drives; a 296-Mbyte streaming tape drive; an integrated disk, tape, Ethernet, and serial line controller; four serial lines;

Ethernet networking; a 105-key keyboard; a three-button mouse; and an optional Q-bus adapter.

For graphics, the 3520 and 3540 have a 19-inch color monitor with a 66 Hz refresh rate and resolution of 1,280 × 1,024, 8 or 24 planes, image buffering of 2K × 2K per plane, 24-bit Z buffering, Gouraud shading, depth cueing, and virtual display lists up to 1 Gbyte.

Operating systems available are Ultrix, with DECwindows, TCP/IP, NFS, VAX PHIGS, VAX C, Fortran, and Pascal, or VMS, with DECwindows, DECnet, Local Area VAX-cluster, and VAX PHIGS.

VAXstation 3520 and 3540 pricing and delivery are scheduled for the second quarter of 1989.

VAXstation 3100: Reader Service 31
3520/3540: Reader Service 32

Chorus speeds Mac nets

Human Devices has announced Chorus, a computation server for networked Apple Macintosh environments. The floor-standing unit contains up to 16 floating-point processors. Each FPP contains a RISC integer processor rated at 5 MIPS, a floating-point unit capable of IEEE-standard double-precision calculations at 2 Mflops, 1 Mbyte of memory, and a Virtual Tree bus interface.

Chorus provides high-level extensions to MPW C, Pascal, and Fortran development environments for applications programmers. According to the company, the software handles development chores associated with parallel processing and networking for the Macintosh.

The Chorus 1, a single-FPP entry-level system, is upgradable to larger configurations. It costs \$9,700. A Chorus 4 configuration with four FPPs costs \$25,500. It includes a dedicated I/O processor with Appletalk and Flashtalk ports and system software. The Chorus 4 system can be upgraded by adding one, two, or three processor boards, each containing an additional four FPPs for \$16,800.

Options include an Ethernet I/O upgrade and an MPW-based programming environment with C, Fortran, and Pascal compilers. A software simulator for \$450 allows programmers to develop for Chorus machines under MPW on unattached Macintosh computers.

Reader Service 33

DECstation based on RISC

Digital Equipment has announced the first in a family of RISC-based workstations, the DECstation 3100. Based on the MIPS Computer Systems chip set, the new workstation reportedly reaches 14 MIPS according to the Dhrystone benchmark, with Linpack performance of 3.7 Mflops single precision and 1.6 Mflops double precision.

The DECstation 3100 runs at a clock speed of 16.67 MHz. It features 8 to 24 Mbytes of memory, a 64-Kbyte instruction cache, a 64-Kbyte write-through data cache, an external SCSI port, an Ethernet controller, a 95-Mbyte streaming tape drive, keyboard and mouse, and a 15- or 19-inch color or monochrome monitor with 1,024 × 864 resolution. The system supports up to two 104-Mbyte 3½-inch SCSI Winchester drives (internally) and up to three 332-Mbyte 5¼-inch SCSI Winchester drives (externally) for maximum storage of 1.2 Gbytes.

Unix software includes a license for Ultrix Workstation Software, which includes the Ultrix-32 operating system, C compiler, X Window System Version 11, X User Interface, DECwindows-based software development tools and applications, TCP/IP, and NFS software. Layered software supported includes a Fortran compiler with VAX extensions and DECnet/Ultrix.

DECstation 3100 prices start at \$11,900 with 8 Mbytes of memory, a 15-inch monochrome monitor, external SCSI port, Thinwire and Ethernet cable



Digital's DECstation 3100 is based on MIPS Computer Systems' RISC chip set and comes in the same desktop package as the VAXstation 3100 system.

interface, C compiler license, Ultrix Workstation Software, mouse, keyboard, and warranty.

The DECstation 3100S server is based on the DECstation 3100 Ultrix workstation. It has a larger memory and disk configuration and is available without a graphics subsystem or monitor. It features 24 Mbytes of memory, 1 Gbyte of disk storage, and switch-selectable Ethernet connections. Scheduled for availability in April, it will cost \$43,400.

**DECstation 3100: Reader Service 34
Server: Reader Service 35**

HP boosts Vectra PC line with laptop and 386-based desk-side models

Hewlett-Packard has announced three additions to its line of HP Vectra personal computers. The HP Vectra RS/20C and RS/25C, both based on the Intel 80386 microprocessor, run at 20 and 25 MHz, respectively. The HP Vectra LS/12 laptop incorporates the Intel 80286 microprocessor. It weighs 10½ pounds without the battery, according to the company.

The new Vectra RS PCs feature a memory subsystem that includes 32 Kbytes of cache memory and an Intel 82385 cache controller.

Using surface-mount technology permits expansion of RAM from 1 Mbyte to 16 Mbytes on the processor boards used in the RS/20C and RS/25C. This leaves eight expansion slots free. An HP human-interface link port allows connection of up to seven HP-HIL devices

without using an expansion slot.

The LS/12 laptop features a 10-inch backlit LCD with 640 × 400 text resolution and a battery pack. The detachable nicad battery weighs about four pounds. An on-screen program tracks remaining battery life.

The laptop comes with 1 Mbyte of RAM, a 3½-inch 1.44 Mbyte floppy disk drive, a choice of 20- or 40-Mbyte hard disk drives, and a 79-key keyboard. It also has HP terminal emulation and HP disk cache software.

The HP Vectra RS/20C comes in three models: the 100e, 150e, and 154e. Model 100e costs \$7,595 for 1 Mbyte of memory, a 5¼-inch 1.2-Mbyte floppy disk drive, and a 103-Mbyte hard disk drive. Model 150e costs \$8,195 with a 155-Mbyte hard disk drive, and Model 154e costs \$10,145 with a 155-Mbyte

hard disk drive and 4 Mbytes of memory.

The HP Vectra RS/25C comes in the same three configurations as the RS/20C at prices of \$10,295 for the Model 100e, \$10,895 for the Model 150e, and \$13,295 for the Model 154e. An additional configuration, Model 304e, has 4 Mbytes of memory, a 5¼-inch 1.2-Mbyte floppy disk drive, and a 310-Mbyte hard disk drive for \$15,695.

The HP Vectra LS/12 laptop Model 24 comes with 1 Mbyte of memory, a 3½-inch 1.44-Mbyte floppy disk drive, and a 20-Mbyte hard disk drive for \$4,879. Model 44 substitutes a 40-Mbyte hard disk drive for \$5,479.

**RS/20C: Reader Service 36
RS/25C: Reader Service 37
LS/12: Reader Service 38**

DECwindows underlies new Digital software

Digital Equipment has based a number of its new software products on DECwindows, including VAX Decision Expert, DECdecision, and DECwrite.

VAX Decision Expert is a graphical shell to assist in the building of expert systems, specifically diagnostic-related expert systems. According to the company, the software does not require developers to be AI professionals. It features a menu-driven interface with graphical formats for entering knowledge as if-then rules, decision trees, and and/or trees. It provides both forward and goal-directed inferencing and automatically generates the end-user interface.

VAX Decision Expert requires the DECwindows interface on any VAXstation for development. It runs under the VMS operating system. Prices range from \$7,400 on the VAXstation 2000 to \$37,740 on the VAX 8978, with deliver-

ies scheduled for March 1989.

DECdecision provides a set of decision support software tools based on the DECwindows graphical interface. It reputedly gives workstation users easy access to distributed data. It features a windowed spreadsheet and a chart-making facility.

DECdecision supports CDA and can share data with CDA applications such as DECwrite. The company recommends using a VAXstation 3100 or higher, with 12 Mbytes and VMS V.5.1. (VAX Rdb/VMS Runtime is included with DECdecision.) Available in May, DECdecision will cost \$1,000 for a single-user license.

DECwrite is a compound document editor for users of the DECwindows environment. It permits users to create and format documents containing text, graphics, images, and applications data. The program, available to both VMS

and Ultrix users, is the first implementation of the CDA architecture.

Under VMS, the company recommends a VAXstation 3100 or higher with a minimum of 8 to 12 Mbytes to operate DECwrite and VMS V.5.1. VAXstations are supported in either client or server mode; all other VAX processors are supported as clients.

Under Ultrix, DECwrite supports the VAXstation 11, 11/GPX, 1000, 3100, 3200, and 3500. VAXstations should have a minimum of 11 to 16 Mbytes to operate DECwrite and UWS V.2.0 (including DECwindows).

Scheduled for availability in May, DECwrite will cost \$1,500 per single-user license.

Decision Expert: **Reader Service 39**
DECdecision: **Reader Service 40**
DECwrite: **Reader Service 41**

eXodus links Macintosh users to X Windows client applications

White Pine Software has announced eXodus, which the company says links Apple Macintosh users to X Window System client applications. The software reportedly turns the Macintosh into an X Window System workstation by acting as a display server.

eXodus conforms to Version 11 of the X Window System. It is compatible with Apple's Multifinder system, which lets the software run in foreground or background mode simultaneously with other applications. For networking, it supports Appletalk, TCP/IP, and

DECnet, as well as Kinetics' TCPPort, and Alisa's TSSnet.

The software runs on a Macintosh with a minimum of 1 Mbyte of memory. eXodus costs \$499.

Reader Service 42

DEC enters PC arena with new IBM-compatible DECstation family

Digital Equipment entered the PC arena with the announcement of a new family of personal computers, the DECstation 210, 316, and 320. The new

IBM-compatible computers run MS-DOS Version 3.3 applications. They come in two configurations: as a basic system with a 3½-inch floppy drive and

VGA graphics adapter, or as a system with VGA graphics adapter, SCSI interface, and hard disk drive.

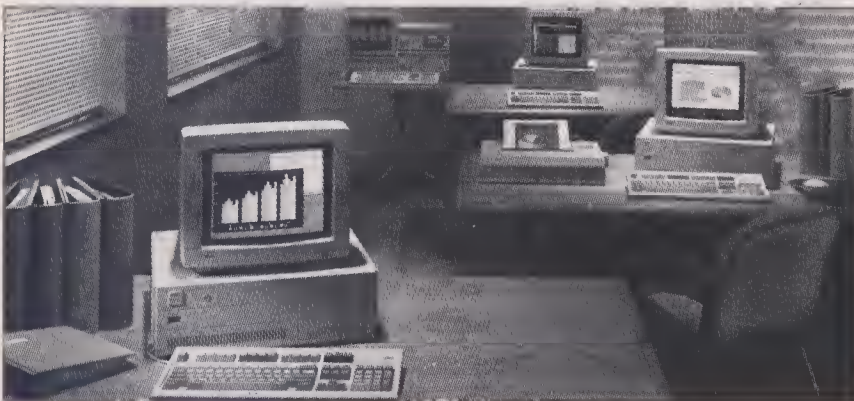
The 210 is based on an Intel 80286 CPU running at 10 MHz. It has 640 Kbytes of RAM, supports four AT and three XT expansion slots, and costs \$2,630. Hard disk configurations come with 40 Mbytes of hard disk storage.

The 316 incorporates an Intel 80386 chip running at 16 MHz. It has 1 Mbyte of RAM, supports six AT and two XT expansion slots, and costs \$3,485.

The 320 is based on an Intel 80386 chip running at 20 MHz. It features 2 Mbytes of RAM, supports six AT and two XT expansion slots, and costs \$4,960.

All three PCs come with a 14-inch monochrome or color monitor, a 101-key keyboard, one dedicated memory option slot, 9-pin serial and parallel ports, and VGA graphics compatibility.

Reader Service 43



DEC's new IBM-compatible PCs fit into a DECnet/PCSA networking environment. They are (front to back) the DECstation 316, 320, and 210. (The PCLAN/Server 2000 is at the back.)

Workstation helps OEMs get Unix-driven RISC to market

Integrated Solutions offers the Advantedge 2000 RISC-based workstation for original equipment manufacturers, value-added resellers, and systems integrators. The new workstation uses MIPS Computer Systems' R2000 RISC chip set. According to the company, the workstation operates at 12 MIPS.

The Advantedge 2000 comes in various configurations, ranging from a stand-alone workstation down to the basic board level. Users can choose one of two Unix operating systems: MIPS' RISC/os (UMIPS) or Integrated Solutions' Dual Universe. The latter has both Unix 4.3 BSD and System V.3.

The company says that the new workstation maintains source code compatibility with its Motorola-based systems.

The workstation features 32 Kbytes of on-board cache memory, 1,280 × 1,024-pixel graphics resolution, an on-board SCSI controller, an Ethernet controller, and an on-board



Integrated Solutions' Advantedge 2000 operates at 12 MIPS, relying on MIPS Computer Systems' R2000 RISC chip set.

80186-compatible I/O processor.

The base system, consisting of a packaged base board and a video board, costs \$12,000. Prices range up to \$24,500 with the addition of peripherals and memory. A system including the

base system, a 16-inch color monitor, a keyboard, a mouse, a 100-Mbyte hard disk drive, and a 40-Mbyte cartridge tape drive costs \$20,375.

Reader Service 44

Security software relies on keystroke dynamics

A software package called Electronic Signature Lock, produced by a company of the same name, assigns users electronic signatures based on their unique keystroke dynamics and typing patterns. It reportedly denies system access to unauthorized users even when they have the proper passwords.

The software analyzes the total typing pattern to identify users. It uses a statistical filtering routine to analyze the patterns and determine the probability of proper identity. Where companies require additional security, they

can decrease the probability of granting unauthorized access by changing the parameters. Alternatively, they could make access easier.

Since the software remembers previous tries to access the system, authorized users face an increased probability of gaining access with more than one attempt, while unauthorized users have less chance of doing so. For example, if an authorized user had a 50 percent chance of gaining access on the first attempt, the probability would rise to 99 percent on the second attempt. If the

probability of an authorized user gaining access on the first try were 99 percent, an unauthorized user would have a one in 10,000 chance for access on the first try. The probability of unauthorized access would fall on subsequent tries.

Prices range from \$500 for a site license for PCs to \$100,000 for a buffered mainframe system with a major network. A demonstration system for PCs costs \$300.

Reader Service 45

Silicon software component adds extensions to VRTX

Metasphere offers Vextra, which the company calls a true silicon software component that adds extensions to the VRTX real-time multitasking executive from Ready Systems. The software reportedly relies on VRTX for basic tasking services and low-level inter-process communication.

Vextra builds on VRTX and VRTX32. The package includes two 32 × 8 EPROMs written in the C programming language, a PC-DOS formatted floppy disk with the Vextra binary

code, and documentation.

Vextra adds 33 new system calls to provide a process model and a suite of memory management services, according to the company. The model provides dynamic process usage through Unix-style management of process birth and death cycles, process parametrization, machine and component exception handling, and built-in event flags. The memory management services provide 255 memory pools deployable in a hierarchy; a choice of three memory alloca-

tion strategies; support for specific block allocation, allocation with timeout, asynchronous allocation, and block chaining; and automatic memory resource ownership tracking and recovery.

Also included is a component development toolkit with runtime system calls.

A research and development license for Vextra costs \$4,950.

Reader Service 46

TeamOne ships Unix-based DBMS software line

TeamOne Systems has begun shipping its object-oriented, Unix-based engineering database management software products. The first products are CM (Configuration Management) and Query, which operate on top of the TeamOne EDM (Engineering Data Management) repository.

According to the company, the EDM software operates as a logical extension to the Unix file system; it does not modify the user's environment. The EDM project repository is an object-oriented database, transparently distributed across the network. It does not replace other database products.

EDM reportedly supports all electronic media. It requires a minimum of 1 Mbyte of disk and 4 Mbytes of

memory.

CM provides capabilities for managing and controlling data and files for engineering product development. The application operates from the EDM database. It supports managing related data, creating and controlling files, merging different versions, tracking changes to data, managing concurrent changes to data, exploring design alternatives, sharing data, setting up test environments, and controlling access to shared data. Like EDM, CM provides a logical extension to Unix, by placing the files under configuration control.

TeamOne Query provides access through the Structured Query Language facility to the EDM project repository. Queries can integrate data from differ-

ent nodes on the network to build integrated queries. The software provides a relational view of the object-oriented repository.

Single copies of TeamOne EDM/CM cost \$2,500, while Query costs \$500. Quantity discounts are available. The company permits prospective users to try the software for up to 30 days without charge, providing free telephone support at the same time. Each site gets a full set of manuals, with additional manuals available.

TeamOne software currently runs on Sun-3 workstations running the Sun O/S 4.0 operating system.

EDM/CM: Reader Service 47
Query: Reader Service 48

VLSI MMU and FPC support Sparc integer unit

The Advanced Products Division of Fujitsu Microelectronics offers VLSI implementations of a 25-MHz memory management unit and floating-point controller for the company's S-25 (MB86901) Sparc RISC processor.

The MMU features 4 Gbytes of virtual address space and 64 Gbytes of physical address space. Up to three levels of page tables support a 4-Kbyte page size. The new MMU also supports large linear mapping with a single page table entry, up to 256 contexts, selective flushing and probing, and 36-bit

addresses. An embedded bus arbiter allows the unit to independently access resources on the shared bus.

The CMOS FPC interfaces between the S-25 Sparc integer unit and the Texas Instruments SN74ACT8847 floating-point processor. It supports a throughput rate of 3.30 Mflops for single-precision operations and 2.70 Mflops for double-precision operations, according to the company. It uses a 64-bit-wide internal data path. The FPC uses 32 working registers, a floating-point state register, and a floating-point

queue.

In quantities of 1,000, the MMU (MB86920) costs \$239 and the FPC (MB86920) costs \$238.

Fujitsu also offers an evaluation board set (BM86901 EB01) consisting of a Sparc VME CPU board, a system memory board, and an optional floating-point module. The price ranges between \$6,000 and \$8,000.

MMU: Reader Service 49
FPC: Reader Service 50
Board set: Reader Service 51

Optical disk organizes medical records

The 3M Document Systems Division has added the Docutron 8000 Medical Information System to its line of electronic document management systems. The new system uses specialized application software to manage medical records within large-volume medical applications.

The Docutron 8000 System handles capture, storage, retrieval, printing, display, and transmission of documents. Its basic components include the host mainframe computer and the database in place within the medical facility; MED/DMS medical application software; a communications network or LAN; and a variety of system controllers and subsystems, such as display terminals, optical disk and magnetic

storage units, scanners, and printers.

The application software merges data from the host or other computer database with documents entered through the 8000 to create a patient information file. This information can be transmitted to workstations on-site or in remote locations for display or printout.

The software also has security levels to restrict access, plus audit trails. Other features include random filing, unit retrieval, access to individual documents or entire medical records, and automatic preadmission chart printing.

Prices for the Docutron 8000 Medical Information System, including hardware and software, start at \$500,000.

Reader Service 52

CTEX brings TEX to PCs

Micro Publishing Systems has released CTEX, a PC-based implementation of TEX version 2.94. CTEX was translated in CWEB format, which according to the company means that the source code is in a form that allows for same-day updates to the TEX language. The company also claims easy porting of the source code to other hardware platforms.

CTEX contains macro packages, including LATEX and AMSTEX, for mathematical equations using specialized symbols.

CTEX costs \$189, with university and government discounts available.

Reader Service 53

PHYSICAL DESIGN WORKSHOP

MODULE GENERATION AND SILICON COMPILATION

April 30, May 1 - 3, 1989

**The Queen Mary
Pier J, Long Beach Harbor
Long Beach, CA 90801**

Workshop Committee

General Chairman

Bryan Preas

Xerox PARC

3333 Coyote Hill Road

Palo Alto, CA 94304

(415) 494-4845

Preas.pa@Xerox.Com

Program Chairman

Daniel Gajski

University of California

Department of Information

and Computer Science

Irvine, CA 92717

(714) 856-4155

Gajski@ICS.UCI.Edu

Benchmark Chairman

Dwight Hill

AT&T Bell Laboratories

600 Mountain Avenue

Murray Hill, NJ 07974

(201) 582-7766

dwight@research.att.com

Registration & Arrangements

Jamie Kirkendall

ACM

11 West 42nd Street

New York, NY 10036

(212) 869-7440

Jamie@ACMVM.BITNET

Workshop Secretary

Lorna Fear

Xerox PARC

3333 Coyote Hill Road

Palo Alto, CA 94304

(415) 494-4813

Fear.pa@Xerox.Com

Fax: (415) 494-4471

The purpose of this Physical Design Workshop is to gain insights into, and to advance the understanding, theory, and practice of, the module generation and silicon compilation aspects of electronic design systems. Session topics will include

- Layout Frameworks for Module Generation and Silicon Compilation
- Analog Module Generation and Silicon Compilation
- Module Generation from Schematics
- Constraint-Driven Optimization
- Description Languages for Behavioral Compilation
- The State of the Art in Silicon Compilation
- Design Models in Silicon Compilation
- Silicon Compilation from VHDL
- Knowledge Representation, Acquisition and Learning in Silicon Compilation

A portion of the workshop will be devoted to the development of module generation and silicon compilation benchmarks that can be used to gain an understanding of the design process and to evaluate various layout systems. To further this goal, a rally will be held to evaluate the benchmarks. Participants are requested to execute the benchmarks and report their results at the workshop.

A copy of the benchmarks can be obtained by writing or sending electronic mail to **Module Generation Benchmarks, Microelectronics Center of North Carolina, P. O. Box 12889, Research Triangle Park, NC 27709, benchmarks@mcnc.org, (919) 248-1915.**

Workshop registration packets can be requested by contacting Jamie Kirkendall by phone ((212) 869-7440), by electronic mail, (Jamie@ACMVM.BITNET), or by mail (ACM, 11 West 42nd St., New York, NY 10036). The packet will include hotel registration as well as workshop registration information. **Requests for registration packets must be received by March 15, 1989. Registration material must be returned to ACM by April 1, 1989. Hotel reservations must be made by March 15, 1989.**

Attendance is limited. In the event too many requests are received, preference will be given to those participating in the program or executing the benchmarks.

SIGDA Travel Grant funds are available to a limited number of participants (students and professors with limited travel funds). Requests should be made to Jim Cohoon, University of Virginia, Thornton Hall, Charlottesville, VA 22903, cohoon@Virginia.EDU, (804) 979-9705.



IEEE COMPUTER SOCIETY

DESIGN AUTOMATION TECHNICAL COMMITTEE



ASSOCIATION FOR COMPUTING MACHINERY

SPECIAL INTEREST GROUP ON DESIGN AUTOMATION



THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.

IC Announcements

Company, Model, Function

Comments

R.S. No.

Analog Devices AD708 Op amp	A dual bipolar op amp featuring 25 μ V max offset voltage and 0.3 μ V/ $^{\circ}$ C max offset voltage drift (S grade) matched to within 25 μ V and 0.3 μ V/ $^{\circ}$ C, respectively. Operates over three temperature ranges. Comes in an 8-pin plastic miniDIP, hermetic ceramic DIP, or TO-99 metal can. Cost (100s): \$2.95 (JN); \$11.50 (S).	120
Burr-Brown OPA675, OPA676 Op amps	Monolithic wide-band (185 MHz) op amps with two differential inputs selectable by external logic signals. Offset voltage of ± 250 μ V. OPA675 is ECL-compatible, OPA676 is TTL-compatible, with input selection speeds of 4 ns and 6 ns, respectively. Both come in 16-pin ceramic DIPs. Cost (1,000s): from \$19.35.	121
Burr-Brown OPA620 Op amp	A monolithic op amp with a gain bandwidth of 200 MHz and a slew rate of 250V/ μ s. Internally compensated for unity gain stability. Comes in 8-pin plastic and ceramic DIPs and 8-lead SOIC packages. Operates over 0-70 $^{\circ}$ C and -55-125 $^{\circ}$ C ranges. Cost (1,000s): starts at \$8.25.	122
Data Translation DT7920 Interface chip	A Micro Channel interface chip. Requires an external data transceiver chip and a POS identification device to function as a complete bus interface. Consumes 100 mA of 5V. Has two independent DMA controllers. Comes in an 84-pin PLCC. Cost (1,000s): \$31.	123
Intel 89C024XE Modem chip set	A 2,400 bps modem chip set with Microcom Networking Protocol Class 5. Provides data compression transfer rates up to 4,800 bps. Supports all modem functions. Consists of a 16-bit application-specific microcontroller and an analog front-end interface. Cost (1,000s): \$40.	124
Hitachi America HD4074308 Microcontroller	A 4-bit microcontroller with 100 instructions. Functions include 8 Kwords of one-time, user-programmable, zero-turnaround-time (ZTAT) EPROM memory, a 4-channel 8-bit A/D converter, two 8-bit timer/counters, 160 $\times$ 4 bits of RAM, 34 I/O pins, a tone generator, and an interrupt controller. Comes in a 42-pin plastic DIP. Cost (1,000s): \$9.99.	125
Integrated Digital Products 10H446 Microprocessor	A 16-bit ECL microprocessor that delivers 40 native MIPS performance with a 120 MHz clock. Includes an on-chip memory management unit, clock oscillator, and memory drivers to drive up to 1 Mbyte of static or dynamic RAM directly. Provides memory bandwidths up to 80 Mbytes/s. Comes in a 149-pin PGA. No prices given.	126
Logic Devices L7C1xxPC SRAM family	Seven SRAMs that retain data at a supply voltage down to 2V. One version of 64 Kbit $\times$ 1 at 20 ns (L7C187PC), four versions of 16 Kbit $\times$ 4 at 25 ns (L7C164PC), and two versions of 8 Kbit $\times$ 8 at 30 ns (L7C185PC). Cost (1,000s): \$16 for L7C187PC, \$11.20 for L7C164PC, \$16.80 for L7C185PC.	127
Matra Design Semiconductor MD series Gate arrays	A family of CMOS gate arrays based on 0.8-micron channel length technology. Gate counts from 250 to 2,500, I/O counts from 31 to 101. Max flip-flop toggle rate of 200 MHz and system clock rate of 125 MHz. Uses two layers of routing, but is programmed with a single metal mask. No prices given.	128
Motorola MCM629xx20 SRAMs	Three 64-Kbit CMOS static RAMs with 20 ns cycle times. Have separate I/O and on-chip registers controlled by a single clock input. MCM6293P20 and MCM6294P20 come in a 28-lead PDIP; MCM6293J20 comes in a 28-lead plastic SOJ. Cost (100s): \$55.16 for MCM629xP20, \$60.70 for MCM6293J20.	129
National Semiconductor DP8420A/21A/22A DRAM controllers	DRAM controllers that interface to microprocessors running up to 25 MHz. DP8422A drives up to 4-Mbit DRAMs and has two access ports. It comes in an 84-pin PLCC for \$23.50 (100s). The single-ported DP8421A drives up to 1-Mbit DRAMs and the DP8420A drives 256-Kbit DRAMs. Both come in 68-pin PLCCs for \$12.50 and \$17.95 (100s), respectively.	130
NMB Technologies AAA1M200 series DRAMs	A series of 1-Mbit DRAMs. Comes in 1 Mbit $\times$ 1 and 256 Kbit $\times$ 4 versions. Max access times of 60, 70, and 80 ns. Also available in enhanced-page and static column mode versions. Comes in plastic DIP, ceramic DIP, ZIP, SOJ, and SIMM packages. Cost (1,000s): around \$20.	131
Sierra Semiconductor SC11024 MAP	A 2,400 bps modem analog peripheral that operates using a 5V power supply. Features power consumption of 65 mW and power-down mode of 12.5 mW typical. Contains all functions except adaptive equalizer. Comes in a 28-pin PLCC and 28-pin DIP. Cost (100s): \$33.35 (PLCC), \$31.75 (DIP).	132

Call for Papers

THE FIRST INTERNATIONAL CONFERENCE ON SYSTEMS INTEGRATION



Headquarters Plaza Hotel, Morristown, New Jersey
April 23-26, 1990

Theme: Integration Technologies with an Emphasis on Computer-Aided Software Engineering, Manufacturing, and Distributed Computer Systems.

This conference focuses on the problems, issues and solutions of integrated system design, implementation and performance. Integration technologies will be emphasized with a focus on computer-aided software engineering, collaborative and distributed systems, and computer integrated manufacturing systems. The conference will provide an international and interdisciplinary forum in which researchers and practitioners can share novel research, engineering development and strategic management experiences. Papers should deal with recent work in theory, design, implementation, utilization and evaluation of integrated systems. Topics to be addressed include, but are not limited to:

Computer-Aided Software Engineering • Knowledge-Based and Expert Systems • Software Development Technologies • Software Bases and Management • Software Specifications • Design and Prototyping • Tools and Methodologies • Computer-Aided Design/Computer-Aided Manufacturing • Computer Integrated Manufacturing • Manufacturing/Engineering Computing • Automated Storage and Retrieval System • Pattern Recognition and Computer Vision • Manufacturing/Engineering Databases • Distributed Processing Systems • Data Communications and Computer Networks • Protocols and Standards • Resource Allocation and Scheduling • Fault-Tolerant Computing • Reliability and Quality Assurance • Modeling and Performance Evaluation/Measurement • Collaborative Systems • Organizational Information Resources • Project Planning and Management • Database Management Systems • Artificial Intelligence Techniques • Man-Machine Interfaces.

Information and Instructions for Authors: Authors are cordially invited to submit original technical papers to the program chairman no later than July 25, 1989. All papers must be in English, typed in double spaced format, and may not exceed 6,000 words. Each submission should provide a cover page containing author(s), affiliation(s) complete address(es), identification of principal author, and telephone number. Also include SIX copies of complete text with a title and abstract. Notice of acceptance will be mailed to the principal author(s) by September 15, 1989. If accepted, the author(s) will prepare the final manuscript in time for inclusion in the conference proceedings and will present the paper at the conference. Authors of accepted papers must sign a copyright release form.

Please send SIX copies of your paper(s) to:
Professor Peter A. Ng, Program Co-Chairperson
Department of Computer and Information Science
New Jersey Institute of Technology
Newark, NJ 07102, U.S.A.

Paper Arrival Deadline: July 25, 1989
Acceptance Notification: September 15, 1989
Final Manuscript Inclusion Deadline:
October 16, 1989

CONFERENCE COMMITTEE

Conference Co-Chairpersons: Larry Seifert, AT&T,
Raymond T. Yeh, Syscorp Inc.

Program Co-Chairpersons: Peter A. Ng, NJIT;
C.V. Ramamoorthy, UC Berkeley

Local Arrangement Chairperson: Fred Poluhovich,
Chemical Bank

Tutorials Chairperson: Frank Shih, NJIT

Vice Chairpersons:

Software Engineering: Leszlo A. Belady, MCC

Manufacturing: Michael Kelly, NJIT and DARPA

Distributed Systems: Erich Neuhold, GMD

Data Management: P. Bruce Berra, Syracuse U.

Information Networking: Robert L. Martin, Bellcore

Communication System: Ming T. (Mike) Liu, Ohio State U.

Automation and Productivity: Ming Leu, NJIT

Computer Languages: Joe Urban, U. Miami

Sponsored by the Institute for Integrated Systems (INIS) and Department of Computer and Information Science at New Jersey Institute of Technology, in cooperation with Gesellschaft fuer Mathematik und Datenverarbeitung (GMD), AT&T, Bell Communications Research (Bellcore), and IEEE Computer Society.

Microsystem Announcements

Company, Model, Function	Comments	R.S. No.
Advanced Micro Devices FAST card Evaluation board	An evaluation board for the Supernet chip set. A plug-in card that converts an IBM PC-AT or compatible into an FDDI node that can be networked to other FDDI nodes. Used with support software, provides a Supernet FDDI development tool set. Cost: \$6,000.	I35
Andrew Network Products Trulynx/3270 PS/PC API API	An applications program interface that provides compatibility with Personal Services/PC. Works with Trulynx/3270 to let laptop users access Personal Services on a mainframe. Provides all five API service categories. Comes in both 3½- and 5½-inch versions. Cost: \$150 for API, \$195 for Trulynx/3270.	I36
Burr-Brown SPVI25 VME board	A multifunction board for VMEbus-based applications. Uses the TMS320C25 processor operating at 25 MHz and two parallel I/O ports under DMA control. Hits data transfer rates up to 4 Mbytes/s. Has 16 Kbytes of zero-wait-state PROM, expandable to 128 Kbytes. Software support available. Cost: \$2,695.	I37
Digital Vision ComputerEyes Video digitizer	A Macintosh version of the gray-scale video digitizer. Supports Macpaint, TIFF, EPS, and PICT2 file formats. Acquires 640 × 480 samples with 256 shades of gray per sample. Plugs into the serial port to connect the Mac with an NTSC or PAL composite video source. Cost: \$249.95.	I38
Electro Design Chipper Developer's tool	Allows users to install MS-DOS applications into EPROMs to create diskless workstations. Creates EPROM-based virtual disks. Provides a bank-select mode to permit up to 16 Mbytes of storage. In bank-select mode, occupies up to 96 Kbytes of a PC's memory map and 4 Kbytes of system memory. Can also be loaded into nonvolatile RAMs. Cost: \$695.	I39
Emulex DCP-186/PC Communications coprocessor	A multiport, multiprotocol communications coprocessor. Incorporates a main module, a daughtercard that supports four RS-232 serial ports and a parallel printer port, and a port termination box with serial and parallel connectors. The main module contains a 10 MHz 80186 microprocessor. Cost: starts at \$1,395.	I40
Ingenion Design Ingenion MTS Test station	A test station for detection of design and construction faults on microprocessor-based printed-circuit boards. Connects to a PC. Compatible with 8-bit microprocessors, including the 8088 and 8086. An intelligent probe identifies logic levels, address and data lines, and can analyze chip-select logic. No price given.	I41
Integrated Device Technology IDT7MB6039/40 Cache module family	A family of data and instruction cache modules for MIPS' R3000 RISC microprocessor. IDT7MB6039 (dual 16K × 60) integrates 30 SRAMs and 20 logic elements. Comes in a 128-pin quad-in-line package in 25, 20, 16, and 12 MHz versions. IDT7MB6040 (dual 16K × 64) is a general-purpose cache. Cost (1,000s): \$875 for 25 MHz IDT7MB6039, \$920 for 25 MHz IDT7MB6040.	I42
Intel Multibus II PC Subsystem PC Subsystem	Consists of a 16 MHz 80386 microprocessor-based CPU board (iSBC 386/PC16); a peripheral companion board (iSBC PCSYS/100) with a hard disk controller, VGA graphics controller, and built-in Multibus II central services; and an adapter board (iSBC PCSYS/900). Cost: \$4,700 for 386/PC16, \$1,295 for PCSYS/100, \$995 for PCSYS/900.	I43
MetraByte PCIP-Scan Scanner/multiplexer board	A reed relay scanner/multiplexer board that plugs into an I/O slot in an IBM PC-XT or AT or compatible. Works with the PCIP family of plug-in instrument boards. Features 16 single-ended or 8 differential channels. Instrument displays occupy one-third of the screen each. Cost: \$325.	I44
Paracom MTM-Sun/xp Motherboard	A 9U, VMEbus-compatible, transputer-based motherboard for Sun systems. Features three 96-pin daughterboard connections. Can provide up to 10 transputers. Contains four T414 or T800 processors, three with 1 Mbyte of single-ported DRAM and one with 4 Mbytes of dual-ported DRAM. Cost: starts at \$12,895.	I45
Radstone Technology 68-33 68030 board	A 68030-based board for VMEbus multiprocessing applications. Has up to 2 Mbytes of quad-ported, no-wait-state SRAM and a VSB interface. Uses a segmented local bus architecture. Takes up to 512 Kbytes of EPROM for application code. Uses the VTC VIC068 VMEbus Interface Controller chip. Cost: starts at \$4,595.	I46
Sonitech International DSP/AIBs Analog interface boards	Digital signal processing analog interface boards to support Motorola's DSP56000ADS Application Development System. DSP/AIB-1 performs acquisition through a serial cable. DSP/AIB-2 fits into a half-card slot of an IBM PC and performs acquisition through the PC backplane or a serial cable. Cost: \$495 for AIB-1, \$695 for AIB-2.	I47



Second Annual IEEE
ASIC Seminar and Exhibit
September 25 - 28, 1989
Rochester Riverside Convention Center
123 East Main Street
Rochester, New York 14604
Sponsored by IEEE Rochester Section
in cooperation with Computer Society



CALL FOR PAPERS, TUTORIALS AND WORKSHOPS

The ASIC Seminar and Exhibit is organized to provide ASIC and system level designers with the knowledge of the tools and techniques required in all phases of ASIC design and implementation. It emphasizes understanding of practical issues of technical details, tradeoffs, and economics of system integration using standard cell and gate array techniques. The conference is intended for ASIC designers, as well as R & D and manufacturing managers focusing on ASIC system implementation. The conference offers: (1) An in-depth introduction to ASIC implementation for system designers, (2) A forum for ASIC practitioners and vendors to share their case design experiences, (3) Executive overviews of ASIC trends, strategy, economics and competitiveness.

HIGHLIGHTS

- Proceedings of Technical Papers
- Design Case Studies
- Tutorial Sessions/Workshops
- Technical Exhibits
- Evening Panel Discussions
- Spouse Program

TUTORIALS (1 to 2 hours in length)

Introduction to ASIC Design	Logic Synthesis
Design Synthesis	Silicon Compilation
Design for Testability	ECL and GaAs Technology
BiCMOS Technology	ASIC manufacturing
Future ASIC Trends for Managers	Selecting a Workstation
Start an ASIC Design Center	

Proposals to organize a tutorial or a session are invited.

WORKSHOPS

Four to eight hour technical workshops covering ASIC design knowledge and skills. Proposals to form these workshops are invited. ASIC industry, as well as universities are encouraged to submit proposals for consideration.

TECHNICAL TOPICS

Technical papers to cover ASIC applications and case studies in the following areas are solicited. The length of presentation shall be 25 minutes, plus 5 minutes for questions.

ASIC DESIGN & SIMULATION
ASIC MEMORIES
ECONOMICS OF ASIC
MANAGING ASIC PROJECTS

DESIGN FOR TESTABILITY
ASIC MANUFACTURING
ASIC TECHNOLOGY SELECTION
ASIC EDUCATION

ASIC APPLICATIONS/CASE STUDIES:

Digital Circuits, Analog Circuits, Parallel Computing, Neural Networks, Bus Controllers, RISC, Electro-Optics Interface, Image Processing, HDTV.

SEND TECHNICAL PROPOSALS TO:

Lynne M. Engelbrecht
Conference Coordinator
170 Mt. Read Blvd.
Rochester, NY 14611
PHONE: (716)-328-2310
FAX: (716)-288-7909

CONFERENCE CHAIRMAN

Jon Edwards

(716) 726-9222
FAX (716) 726-5645

TECHNICAL PROGRAM CHAIRMAN

Ken Hsu

(716) 588-4123
FAX (716) 477-4947

EXHIBITS CHAIRMAN

Pete Parslow

(716) 288-7900
FAX (716) 288-7909

INSTRUCTIONS:

Five copies of a 400 - 500 word of summary, and a 50 word abstract, typed single-spaced on 8½ x 11 white paper. Must be received for paper selection purposes by **March 15, 1989**. Author's name, affiliation, complete mailing address, telephone number and tele/fax **MUST** appear on the summary.

CONFERENCES

Editor: Edmund L. Gallizzi, Computer Science Dept., Eckerd College, St. Petersburg, FL 33733; (813) 864-8272; Compmail, e.gallizzi

Symposium keynoter Browne stresses importance of parallelism

Gerald M. Karam, Carleton University

J.C. Browne of the University of Texas at Austin presented a unified approach to parallelism when he delivered the keynote address at the first International Symposium on Databases in Parallel and Distributed Systems. The event was held December 5-7 in Austin, Texas.

Joseph Urban of the University of Miami was general chair of the conference, sponsored by the IEEE Computer Society's Technical Committee on Data Engineering and ACM's SIGArch. Won Kim of MCC, Sushil Jajodia of George Mason University, and Avi Silberschatz of the University of Texas at Austin were the program co-chairs.

Browne said that parallelism is a central conceptual model in computer science and that researchers in the various subfields often regard parallelism in their domain as somehow different. This leads to constant "reinvention of the wheel" and the "not-invented-here" syndrome.

At the heart of Browne's work is an abstract model of parallel computing based on a set of components coordinating their activities to meet the specification for a computation. Computation is a set of states that progress by successive transformations. Firing rules and dependency relations are the key elements of the transformation process.

Browne observed that these elements are the roots of the major differences between various views of parallel computing. Browne's solution is a declarative, component-based, directed-graph programming language that expresses abstract parallelism.

His talk provoked debates that culminated in a panel discussion entitled "Parallelism in Databases: What, Why, and Whither." The panelists presented several perspectives on the role and potential for parallel databases.

Chaired by John Carlis of the University of Minnesota, the panel featured

Roger King of the University of Colorado at Boulder, Jaideep Srivastava of the University of Minnesota, and Justin Rattner of Intel Scientific.

After the panelists made their position presentations, comments and debate from the floor drew out such questions as "Are there concurrency issues unique to databases?" and "Can compilers do all the work so that the specification of concurrency in database systems is hidden from the user?" In addition, Carlis posed the question: Will database management systems simply migrate into basic operating system functions (that already manage concurrent operations) and cease to exist altogether? Hector Garcia-Molina of Princeton University remarked that perhaps the reverse situation will occur.

In his closing remarks, Carlis chal-

lenged the audience to examine these issues and specifically consider how parallel database research will be driven by concurrent hardware and programming languages.

According to Kim, the symposium brought together papers representing key work in load balancing, impacts of richer data models on DBMS architectures, performance modeling, query optimization/processing, and heterogeneous distributed databases.

The symposium proceedings can be ordered by contacting the Computer Society Press, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578, phone (800) CS-BOOKS [in California, dial (714) 821-8380] and specifying order No. 893. The IEEE and Computer Society-member price is \$32, and the nonmember price is \$64.

IEEE workshop to explore image sequence motion

Brian G. Schunck, University of Michigan

Slated March 20-22 in Irvine, California, the IEEE Workshop on Visual Motion will bring together researchers from computer vision, artificial intelligence, and psychophysics to discuss current work on the representation and analysis of motion in image sequences.

The IEEE Computer Society is sponsoring the event, with Brian Schunck of the University of Michigan serving as the general chair. Ramesh Jain of the University of Michigan and Ellen Hildreth of the Artificial Intelligence Laboratory will be the program chairs.

Papers will be presented on all aspects of the analysis of motion in human and machine vision. The papers have been selected to reflect the state of the art in computational theories of motion perception, provide recent results from psychophysical experi-

ments, and discuss attempts to incorporate motion algorithms in practical applications.

Paper topics will include object tracking, estimation of the parameters of object motion, motion constraint equations, estimation of the image flow velocity field, motion correspondence, psychophysical experiments on motion perceptions, structure from motion, visual navigation, and object discrimination in dynamic scenes.

The number of presentations will be limited to maximize the amount of time available for discussion.

Further information can be obtained by contacting Brian G. Schunck, Artificial Intelligence Laboratory, University of Michigan, Ann Arbor, MI 48109-2122, phone (313) 747-1803.

Funt, Ho presented Marr Award at ICCV 88

Laurie White, Eckerd College

Brian V. Funt and Jian Ho of Simon Fraser University were presented the Marr Award for delivering the best paper at the second International Conference on Computer Vision, held near Tampa, Florida, December 5-8.

Ruzena Bajcsy of the University of Pennsylvania and Shimon Ullman of the Massachusetts Institute of Technology cochaired ICCV 88, sponsored by the IEEE Computer Society's Technical Committee on Pattern Analysis and Machine Intelligence (PAMI).

Funt and Ho's paper was entitled "Color from Black and White." Chromatic aberration is often considered an undesirable distortion. In their paper, Funt and Ho showed how chromatic aberration can be beneficial in providing spectral information that can solve the color constancy problem.

The Marr Award is named after the late David Marr, a prominent researcher in computational modeling of human vision.

A wide variety of areas that contribute to computer vision, including color, shape, motion, object recognition, features, and techniques, were presented in papers at ICCV 88 ranging from the highly theoretical to reports on full implementations, often complete with impressive films and slides of the results.

Marr Award runner-up honorable mentions went to Vishvjit Nalwa of AT&T Bell Laboratories for a paper entitled "Representing Oriented Piecewise C^2 Surfaces," presented in the session on shape; to Alan Yuille of Harvard University and Norberto Grzywacz of Witaker College for "The Motion Coherence Theory," from one of the motion sessions; and to David Lowe of the University of British Columbia for "Organization of Smooth Image Curves at Multiple Scales," from the features session.

Nalwa's paper, presented by T.E. Boulton of Columbia University, focused on finding an appropriate formalism for shape representation. After discussing some of the better known existing representations, Nalwa proposed representing surfaces on a Gaussian sphere. This would have the advantages of being convenient, locally generative, and unambiguous.

In another theoretical paper, Yuille and Grzywacz examined how psychophysical developments in coherence relate to computer vision. Because coherent motion, the tendency of nearby objects to move together, is not

fully addressed within existing theories of visual motion, Yuille and Grzywacz proposed the motion-coherence theorem. Their paper described the theory for both long- and short-range motion, compared it to other theories, and demonstrated how motion coherence provides a solution to the aperture problem.

Lowe's paper was more application-oriented, describing methods for multiscale analysis of curves. He combined the tasks of smoothing and segmenting curves at multiple levels of smoothness. Additionally, he dealt with the problem of curve shrinkage that can result from traditional Gaussian filtering. These techniques should provide descriptions of images that are stable across different viewpoints and conditions for appli-

cations such as model-based vision and perceptual organization.

Also at the conference, a Computer Society Certificate of Appreciation was presented to J.K. Aggarwal of the University of Texas at Austin for excellence as guest coeditor of the first issue of *IEEE Expert* with Richard O. Duda. The issue was published in the spring of 1986.

The conference proceedings can be obtained by contacting the Computer Society Press, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578, phone (800) CS-BOOKS [in California, dial (714) 821-8380] and specifying order No. 883. The IEEE and Computer Society-member price is \$55, and the nonmember price is \$110.

Kurzweil, Tedesco, Kempf to keynote conference on use of artificial intelligence technology

Christopher Locke, Text Systems Consulting

Ray Kurzweil, Len Tedesco, and Karl Kempf will be the keynote speakers when the fifth IEEE Conference on Artificial Intelligence Applications convenes March 6-10 in Miami.

Elaine Kant of Schlumberger will serve as general chair of the IEEE Computer Society-sponsored event, with Mark Fox and Roy Maxion of Carnegie Mellon University the program cochairs.

The CAIA keynoters reflect a practical orientation. Kurzweil, chairman of Kurzweil Applied Intelligence, will deliver the opening address, "Artificial Intelligence: Real Technology for Real People."

Tedesco, supervisor of Ford's Service Systems Design section, will focus on "Automotive Diagnostic Equipment Applications of Expert Systems Technology." He will discuss his company's service bay diagnostic systems project.

Kempf, principal scientist at Intel's Knowledge Applications Laboratory, will talk on "Manufacturing Planning and Scheduling: Where We Are and Where We Need To Be." He will describe the problems and opportunities associated with the use of AI in manufacturing scheduling and process planning.

CAIA will focus on the pragmatic deployment of expert systems to enhance productivity in real-world com-

mercial and industrial applications. This conference traditionally concerns concrete issues surrounding the use of the technology while providing a supporting theoretical component.

The paper sessions will include tracks on design and reasoning, intelligent interfaces, tools and architectures, diagnosis, concept learning, knowledge acquisition, and planning and scheduling.

The papers will cover a broad range of application issues, including the use of knowledge-based systems in communication network configuration, industrial machine simulation, 3D object recognition, automatic classification for information retrieval, mixed-initiative inferencing, real-time signal monitoring, fault diagnosis in digital circuits, semantic clustering in case-based reasoning, machine learning and prediction, real-time dynamic scheduling, and planning for cooperative work groups.

March 6 and 7, the two days preceding the conference proper, will be devoted to tutorials.

Additional details, including registration, travel, and accommodation information, can be obtained by contacting the Computer Society Conference Department, 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-0101.

“Digital’s Database Systems Groups are on the route to the 1990s”.

We're committed to moving forward towards future generations of Digital products.

Our winning organization is offering exceptional and diverse technical challenges to database engineers.

Move into the fast lane of database development with Digital in Nashua, New Hampshire or Colorado Springs, Colorado.

Database Product Development

Colorado Springs, Colorado & Nashua, New Hampshire

We develop internals of database management systems like Rdb/VMS, DBMS, Database Distributor and SQL. New areas of attention include extended relational models, knowledge-based systems and object-oriented database management systems. We are also developing the next generation of high performance, extended relational database systems using state-of-the-art technologies. These positions require an MS/BSCS or equivalent and several years' experience in database internals, as well as expertise in:

- Distributed Database (distributed query optimization, concurrency control, distributed transaction management)
- Database Languages
- Multi-vendor Database Interoperability
- Relational Database Development
- Object-oriented Languages and Database Products

DATABASE SUPERVISORS/PROJECT LEADERS
PRINCIPAL SOFTWARE ENGINEERS

Database Performance Analysis

Colorado Springs, Colorado & Nashua, New Hampshire

We provide all predictive performance modeling support for all future database products, involving predictive analytical modeling as well as simulation modeling. We also conduct performance testing and benchmarking for current products.

MATH MODELER — Experience in analytic queueing network methodology and solution algorithms, and ability to map complex systems onto Markovian and/or queueing models.

SIMULATION ENGINEERS — Performance predictions by running event and/or process-driven simulations of designs.

PERFORMANCE TEST ENGINEERS — Run large-scale benchmarking lab, utilizing multi-processor, multi-systems configurations.

Database Architecture

Colorado Springs

We are responsible for the industry's most advanced and versatile relational database architecture, including interfaces and languages design and compliance verification. DATABASE ARCHITECTS — All openings require at least 3 years' applicable technical experience.

Move into the database development fast lane. Send your resume now, indicating your geographic preference to:

For openings in Nashua, NH:

Chris Baker
Digital Equipment Corporation
Department 0201 9133
110 Spit Brook Road
Nashua, New Hampshire 03062

For openings in Colorado Springs, CO:

Mike Johnston
Digital Equipment Corporation
Department 0201 9133
1175 Chapel Hills Drive
Colorado Springs, Colorado 80920

We are an affirmative action employer

**Digital
has
it
now.**

CALENDAR

February 1989

Ninth Int'l Assoc. for Computer Operations Conf., Feb. 27-Mar. 2, San Diego, Calif. Contact AFCOM, Orange, Calif., phone (714) 997-7966.

Compcon Spring 89, Feb. 27-Mar. 3, San Francisco. Contact Kenichi Miura, Computational Research Dept., MS B2-7, Fujitsu America, 3055 Orchard Dr., San Jose, CA 95134-2017, phone (409) 432-1300, ext. 5408 or 5723; or Roy Lee, Sandia National Labs, PO Box 969, Livermore, CA 94551, phone (415) 294-2127.

Joint AdaJUG/ACM SIGAda Meeting, Feb. 27-Mar. 3, Santa Clara, Calif. Contact Danieli and O'Keefe at (800) 522-0300.

Securicom 89, Seventh Worldwide Congress on Computer and Communications Security and Protection, Feb. 28-Mar. 3, Paris. Contact SEDEP, 8, rue de la Michodière, 75002, Paris, France, phone 33 (1) 47-42-40-30.

March 1989

First Oregon Workshop on Software Metrics, Mar. 1-2, Portland, Ore. Cosponsors: Oregon Center for Advanced Technology Education, Portland State Univ., Oregon State Univ. Contact Warren Harrison, Computer Science Dept., Portland State Univ., PO Box 751, Portland, OR 97207-0751, phone (503) 464-3108.

Int'l Conf. on Expert Systems for Development, Mar. 1-3, Kathmandu, Nepal. Cosponsors: Asian Inst. of Technology, Royal Nepal Academy of Science and Technology. Contact Division of Computer Science, AIT, GPO Box 2754, Bangkok 10501, Thailand.

89 Computer Expo for Consulting Engineers, Mar. 2, Chicago. Sponsor: Structural Engineers Assoc. of Illinois. Contact Timothy J. Kilberg, Envirodyne Engineers, 168 N. Clinton St., Chicago, IL 60606, phone (312) 648-1700.

HCCA 4, Fourth Conf. on Hypercube Concurrent Computers and Applications, Mar. 6-8, Monterey, Calif. Sponsors: US Dept. of Energy et al. Contact John Gustafson, Div. 1413, Sandia National Labs, PO Box 5800, Albuquerque, NM 87185.

EMC 89, Eighth Zurich Symp. and Technical Exhibition on Electromagnetic Compatibility, Mar. 6-9, Zurich. Sponsor: Swiss Electrotechnical Assoc. Contact T. Dvorak, EMC 89, ETH Zentrum-IKT, CH-8092, Zurich, Switzerland, phone 41 (1) 256-2790.

 Conferences that the IEEE Computer Society sponsors or participates in are indicated by the Computer Society logo; additional conference sponsors are also listed. Other conferences of interest to our readers are included, as well.

For inclusion in Call for Papers or Calendar, submit information **six weeks before the month of publication** (i.e., for the **April 1989** issue, send information for receipt by **Feb. 15, 1988**) to Chuck Governale, Calendar Dept., *Computer*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.

CAIA 89, Fifth Int'l Conf. on Artificial Intelligence Applications, Mar. 6-10, Miami, Fla. Contact Mark S. Fox, Robotics Inst., Carnegie Mellon Univ., Pittsburgh, PA 15213, phone (412) 268-5016; or Elaine Kant, Schlumberger-Doll Research, Old Quarry Rd., Ridgefield, CT 06877-4108, phone (203) 431-5516.

Seventh IEEE Computer Fair, Mar. 8-10, Huntsville, Ala. Joint sponsors: Huntsville IEEE Section, Huntsville IEEE Computer Society. Contact Connie Harbison, TotalCom, 1115 Church St., Suite H, Huntsville, AL, phone (205) 534-6383.

NCAT, Seventh National Conf. on Ada Technology, Mar. 13-16, Atlantic City, NJ. Contact NCAT, US Army Communications—Electronics Command, Attn.: AMSEL-RD-SE-CRM (Kay Trezza), Fort Monmouth, NJ 07703-5000, phone (201) 532-1898.

Tapsoft 89, Joint Conf. on Theory and Practice of Software Development, Mar. 13-17, Barcelona, Spain. Sponsor: European Assoc. of Theoretical Computer Science. Contact Pere Botella, Facultat d'Informàtica, Pau Gargallo, 5, 08028 Barcelona, Spain, phone 34 (3) 333-8308.

Second Teamworkers Int'l Conf., Mar. 14-17, Hilton Head, S.C. Sponsor: Teamworkers User Group. Contact Karen Chiacu, Cadre Technologies, 222 Richmond St., Providence, RI 02903, phone (401) 351-5950.

Second Computer Virus Clinic, Mar. 15, New York City. Sponsor: Data Processing Management Assoc. Contact Conf. Coordinator, Box 894, Wall St. Station, New York, NY 10268, phone (800) 835-2246, ext. 190.

15th Computer Fair, Mar. 15-16, Seattle, Wash. Sponsor: Univ. of Washington. Contact Sheryl Blix Burghshahler, Desktop Computing Services, Univ. of Washington, FK-10, Seattle, WA 98195, phone (206) 543-0683.

Computing: Policy and Social Issues National Conf., Mar. 16-18, Chattanooga, Tenn. Sponsor: Society for the Psychological

Study of Social Issues. Contact Continuing Education Div., Univ. of Tennessee at Chattanooga, 119 Race Hall, 615 McCallie Ave., Chattanooga, TN 37403-2598, phone (615) 755-4344.

Workshop on Visual Motion, Mar. 20-22, Irvine, Calif. Contact Ellen Hil-dreth, Artificial Intelligence Lab, 545 Technology Square, Cambridge, MA 02139; or Brian G. Schunck, Electrical Engineering and Computer Science Dept., 1301 Beal St., Univ. of Michigan, Ann Arbor, MI 48109-2122, phone (313) 747-1803.

Mid-Lantic Electronics Show, Mar. 21-22, Valley Forge, Pa. Sponsor: Electronic Representatives Assoc. Contact Judith Ginsberg, 4113 Barberry Dr., Lafayette Hill, PA 19444, phone (215) 828-2271.

First Neural Networks in the Real World Conf., Mar. 22-23, San Jose, Calif. Sponsor: Int'l Planning Information. Contact Murray Disman, IPI, 465 Convention Way #1, Redwood City, CA 94063, phone (415) 364-9040.

PCCC 89, Phoenix Conf. on Computers and Communications, Mar. 22-24, Scottsdale, Ariz. Cosponsor: Arizona State Univ. Contact T.D. Regulinski, Reliability Training Inst., PO Box 275, Avondale, AZ 85323, phone (602) 264-2335.

AISIG 89, Artificial Intelligence Systems in Government Conf., Mar. 27-31, Washington, DC. Cosponsors: George Washington Univ., Mitre Corp. Contact Jude Franklin, Government Information Systems, 1500 Planning Research Dr., McLean, VA 22102, phone (703) 556-1990; or IEEE Computer Society, 1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-1013.

First Conf. on Innovative Applications of Artificial Intelligence, Mar. 28-30, Stanford, Calif. Sponsor: American Assoc. for Artificial Intelligence. Contact AAAI, 445 Burgess Dr., Menlo Park, CA 94025, phone (415) 328-3123.

Eastern Multiconference, Mar. 28-31, Tampa, Fla. Sponsor: Society for Computer

Simulation. Contact SCS, PO Box 17900, San Diego, CA 92117-7900, phone (619) 277-3888; or Allan Rutan, Radar Systems Lab, Raytheon, Wayland, MA 01778, phone (617) 440-5088.

Third Parallel Processing Symp., Mar. 29-31, Fullerton, Calif. Cosponsors: California State Univ., Fullerton et al. Contact Larry Canter, Computer Systems Approach, Inc., 1140 S. Raymond Ave., Suite B, Fullerton, CA 92631, phone (714) 738-3414.

Eighth Symp. on Principles of Database Systems, Mar. 29-31, Philadelphia. Sponsor: ACM. Contact Avi Silberschatz, Computer Science Dept., Univ. of Texas at Austin, Austin, TX 78712.

Workshop on Applied Computing 89, Mar. 30-31, Stillwater, Okla. Cosponsors: National Science Foundation et al. Contact Donald D. Fisher, Oklahoma State Univ., CIS, Stillwater, OK 74078, phone (405) 624-5668.

CMC 89, Colorado Microelectronics Conf., Mar. 30-31, Colorado Springs, Colo. Contact Conf. Secretary, CMC 89, Microelectronics Research Labs, Univ. of Colorado, Box 7150, Colorado Springs, CO 90933-7150, phone (719) 593-3488.

Western Educational Computing Conf., Mar. 30-31, Santa Cruz, Calif. Sponsor: California Educational Computing Consortium. Contact Judah Rosenwald, Extended Education, NAD 153, San Francisco State Univ., 1600 Holloway, San Francisco, CA 94132, phone (415) 338-1212.

April 1989

Second National Conf. on Telecommunications, Apr. 2-5, York, England. Sponsor: Institution of Electrical Engineers. Contact IEE Conf. Services, Savoy Pl., London WC2R 0BL, UK, phone 44 (1) 240-1871, ext. 222.

ASPLOS III, Third Int'l Conf. on Architectural Support for Programming Languages and Operating Systems, Apr. 3-6, Boston. Cosponsor: ACM. Contact Joel Emer, DEC/MIT, 545 Technology Square (NE43-503), Cambridge, MA 02139, phone (617) 253-7341.

Working Conf. on Visual Database Systems, Apr. 3-7, Tokyo. Cosponsors: IFIP, IPSJ. Contact IFIP TC-2 Working Conf., Tosiya L. Kunii, Information Science Dept., Faculty of Science, Univ. of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan, phone 81 (3) 812-2111, ext. 4116.

Third Eurographics Workshop on Intelligent CAD Systems, Apr. 3-7, Texel, The Netherlands. Sponsor: Center for Mathematics and Computer Science. Contact Marja Hegt, CMCS, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, phone 31 (20) 592-4058.

FLAIRS 89, Second Florida Artificial Intelligence Research Symp., Apr. 3-7, Orlando, Fla. Contact Avelino J. Gonzalez, Computer Engineering Dept., Univ. of Central Florida, Orlando, FL 32816, phone (407) 281-5027.

Int'l Symp. on Database Systems for Advanced Applications, Apr. 10-12, Seoul, Korea. Cosponsors: IPSJ, Korean Information Science Society. Contact Sukho Lee, Computer Engineering Dept., Seoul National Univ., Sinlim-Dong, Gwanak-ku, Seoul 151, Korea, phone 82 (2) 886-0101.

MIV 89, Int'l Workshop on Industrial Applications of Machine Intelligence and Vision, Apr. 10-12, Tokyo. Cosponsors: IEEE et al. Contact Mitsuru Ishizuka, Inst. of Industrial Science, Univ. of Tokyo, 7-22-1, Roppongi, Minato-ku, Tokyo, 106, Japan, phone 81 (03) 470-5389.

First Hong Kong Int'l Computer Conf., Apr. 10-14, Hong Kong. Sponsor: American Federation of Information Processing Societies et al. Contact George R. Eggert, AFIPS, 2200E Devon Ave., Suite 268, Des Plaines, IL 60018, phone (312) 299-4227; or Alex Tzang, Suite 705 East Town Bldg., 41 Lockhart Rd., Hong Kong, phone 852 (5) 286-136.

1989 IEEE VLSI Test Workshop, Apr. 11-13, Atlantic City, NJ. Sponsors: IEEE Computer Society Test Technology Committee, IEEE Philadelphia Section. Contact Wesley E. Radcliffe, IBM-CTD, B/321-5E1, D/277, East Fishkill Facility, Hopewell Junction, NY 12533, phone (914) 894-4346.

ETC 89, First European Test Conf., Apr. 12-14, Paris. Cosponsor: Societe des Electriciens et des Electroniciens. Contact Roger Cogonen, 36 Ave. Jean Janurs, 95230 Soisy Sous, Montmorency, France, phone 33 (1) 39-89-03-46; or Colin Maunder, British Telecom Research Labs, Martlesham Heath, Ipswich, Suffolk IP5 7RE, phone 44 (473) 642-706.

NCGA 89, Apr. 17-20, Philadelphia. Sponsor: National Computer Graphics Assoc. Contact NCGA, 2722 Merrilee Dr., Suite 200, Fairfax, VA 22031, phone (800) 225-6242.

First Symp. on Parallel and Distributed Processing, Apr. 20-21, Dallas. Contact Behrooz Shirazi, Computer Science Dept., Southern Methodist Univ., Dallas, TX 75275, phone (214) 692-2874.

Multimedia 89, Second ComSoc Int'l Multimedia Communications Workshop, Apr. 20-23, Ottawa, Canada. Cosponsors: IEEE et al. Contact Ottawa Carleton Research Inst., 300 March Rd., Suite 204, Kanata, Ont., Canada K2K 2E2, phone (613) 592-8160.

Infocom 89, Conf. on Computer Communications, Apr. 23-27, Ottawa, Canada. Contact Celia Desmond, Telecom Canada, 438 Bay St., FL5S (C5), South

Tower, Toronto, Ont., Canada, M5G 2E1, phone (416) 581-2318.

Vision 89, Apr. 25-27, Chicago. Sponsor: Society of Manufacturing Engineers. Contact Maria Nowakowski, SME, 1 SME Dr., PO Box 930, Dearborn, MI 48121, phone (313) 271-1500, ext. 376.

Third Workshop on Empirical Studies of Programmers, Apr. 29-30, Austin, Tex. Sponsor: Foundation for the Empirical Studies of Programmers. Contact Gary Olson, Cognitive Science and Machine Intelligence Lab, Univ. of Michigan, Ann Arbor, MI 48109-1234, phone (313) 747-4948; or Elliot Soloway, Univ. of Michigan, EECS Dept., 1101 Beal Ave., Ann Arbor, MI 48109, phone (313) 936-1562.

Physical Design Workshop, Apr. 30-May 3, Long Beach, Calif. Contact Bryan Preas, Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA 94304, phone (415) 494-4845.

CHI 89, Conf. on Human Factors in Computing Systems, Apr. 30-May 4, Austin, Tex. Cosponsors: ACM, Human Factors Society. Contact Claudia Raun or Bill Curtis, MCC, PO Box 200195, Austin, TX 78720, phone (512) 338-3798.

34th Int'l Instrumentation Symp., Apr. 30-May 4, Orlando, Fla. Sponsor: Instrument Society of America. Contact Frederick A. Kern, PO Box 65, Seaford, VA 23696, phone (804) 865-3269.

May 1989

1989 IEEE Symp. on Research in Security and Privacy, May 1-3, Oakland, Calif. Contact Terry V. Benzel, Trusted Information Systems, 11340 Olympic Blvd., Suite 265, Los Angeles, CA 90064, phone (213) 477-5828.

Physical Design Workshop, May 1-3, Newport Beach, Calif. Cosponsor: ACM. Contact Bryan Preas, Xerox, Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304, phone (415) 494-4845.

Third AI Research in Environmental Science Workshop, May 2-4, Washington, DC. Contact William R. Moninger, Environmental Research Labs, NOAA, R/E2, 325 Broadway, Boulder, CO 80803, phone (303) 497-6435.

Sixth Canadian Symp. on Instructional Technology, May 3-5, Halifax, N.S., Canada. Sponsor: National Research Council Canada. Contact Laurier Forget, CSIT, Conf. Services, NRCC, Ottawa, Ont. K1A 0R6, Canada, phone (613) 993-9009.

20th Pittsburgh Conf. on Modeling and Simulation, May 4-5, Pittsburgh, Pa. Sponsor: Univ. of Pittsburgh. Contact William G. Vogt or Marlin H. Mickle, 348 Benedum Engineering Hall, Univ. of Pittsburgh, Pittsburgh, PA 15261.

STA 5, Fifth Structured Techniques Assoc. Conf., May 8-11, Chicago. Sponsors: STA, ACM. Contact STA, c/o Robert Binder Systems Consulting, Inc., 3 First National Plaza, Suite 1400, Chicago, IL 60602.

34th Int'l SAMPE Symp., May 8-11, Reno, Nev. Sponsor: Society for the Advancement of Material and Process Engineering. Contact Marge Smith, SAMPE, PO Box 2459, 843 W. Glentana, Covina, CA 91722, phone (818) 331-0616.

 **CompEuro 89, Int'l Conf. on VLSI and Computer Peripherals, May 8-12**, Hamburg. Cosponsors: Gesellschaft für Informatik et al. Contact Walter E. Proebster, IBM Lab, PO Box 1380, D-7030 Boeblingen, Schönaicher Str. 220, Federal Republic of Germany, phone 49 (70) 3116-3929.

ICCAL 89, Second Int'l Conf. on Computer-Assisted Learning, May 9-11, Dallas. Sponsor: Computer Learning Research Center, Univ. of Texas at Dallas. Contact Janet Harris, Center for Continuing Education, Univ. of Texas at Dallas, PO Box 830688, MS CN 1.1, Richardson, TX 75083-0688.

Robots 13, May 9-12, Gaithersburg, Md. Sponsor: SME. Contact Rebecca Alsup, SME, 1 SME Dr., Dearborn, MI 48121, phone (313) 271-1500, ext. 358.

CCC 89, Second Hungarian Custom Circuit Conf., May 10-12, Szeged, Hungary. Cosponsors: Scientific Society of Measurement and Automation (MATE). Contact MATE Secretariat, 1055 Budapest, Kossuth L. tér 6-8, Hungary, phone (1) 531-406.

 **Sixth IEEE Workshop on Real-Time Operating Systems and Software, May 11-12**, Pittsburgh. Cosponsor: Carnegie Mellon Univ. Contact Andre van Tilborg, Office of Naval Research, 800 N. Quincy St., Arlington, VA 22217-5000, phone (202) 696-4302.

AAMSI Congress 89, May 11-13, San Francisco. Sponsor: American Assoc. for Medical Systems and Informatics. Contact AAMSI, Suite 700, 1101 Connecticut Ave. NW, Washington, DC 20036, phone (202) 857-1189.

First Int'l Workshop on Human and Machine Cognition, May 11-13, Santa Rosa Island, Fla. Contact Ken Ford, Computer Science Div., Univ. of West Florida, Pensacola, FL 32514, phone (904) 474-2551.

Int'l Conf. on Robotics and Automation, May 14-19, Scottsdale, Ariz. Sponsor: IEEE. Contact Harry Hayman, PO Box 3216, Silver Spring, MD 20901, phone (301) 434-1990 or (407) 483-3037.

 **11th Int'l Conf. on Software Engineering, May 15-18**, Pittsburgh. Cosponsor: ACM. Contact Larry Druffel, Software Engineering Inst., Carnegie Mellon Univ., Pittsburgh, PA 15233, phone (412) 963-1318 or 268-7740; or IEEE Computer Society,

1730 Massachusetts Ave. NW, Washington, DC 20036-1903, phone (202) 371-0101.

Sixth Conf. on Real-Time Applications in Nuclear, Particle, and Plasma Physics, May 15-18, Williamsburg, Va. Sponsors: IEEE et al. Contact Roy Whitney, 12000 Jefferson Ave., Newport News, VA 23606, phone (804) 249-7536.

18th Mumps Users Group Annual Meeting, May 15-19, Seattle. Contact Mumps Users Group, 4321 Hartwick Rd., Suite 510, College Park, MD 20740, phone (301) 779-6555.

SID 89, Society for Information Display Int'l Symp., Seminar, and Exhibition, May 15-19, Baltimore. Contact Society for Information Display, c/o Palisades Inst. for Research Services, 201 Varick St., Rm. 1140, New York, NY 10014, phone (212) 620-3375.

 **Int'l Symp. on VLSI Technology, Systems, and Applications, May 17-19**, Taipei, Taiwan. Cosponsors: Republic of China National Science Council, Industrial Technology Research Inst. Contact Alice M. Chiang, MIT, Lincoln Lab, Lexington, MA 02173-0073, phone (617) 981-4629.

Chapel Hill Workshop on Volume Visualization, May 18-19, Chapel Hill, N.C. Sponsor: Univ. of North Carolina, Molecular Graphics Society. Contact Frederick P. Brooks, Jr., Computer Science Dept., Univ. of North Carolina, Chapel Hill, NC 27599-3175.

 **Fifth Int'l Workshop on Software Specification and Design, May 19-20**, Pittsburgh. Cosponsor: ACM. Contact Sol J. Greenspan, GTE Labs, 40 Sylvan Rd., Waltham, MA 02254, phone (617) 466-2962; or Colin Potts, MCC, 9390 Research Blvd., Kaleido II Bldg., Austin, TX 78759, phone (512) 338-3629.

 **First IEEE Symp. on Parallel and Distributed Processing, May 22-23**, Dallas. Sponsor: Dallas Section of the IEEE Computer Society. Contact Mark Shadwens, Information Technologies Lab, Texas Instruments, PO Box 655474, MS 238, Dallas, TX 75265, or call Behrooz Shirazi at (214) 692-2874.

AMAST, Int'l Conf. on Algebraic Methodology and Software Technology, May 22-24, Iowa City, Iowa. Contact Eugene Madison or Teodor Rus, Computer Science and Mathematics Dept., Univ. of Iowa, Iowa City, IA 52242, phone (319) 335-0742 or 0694.

Sixth Int'l Conf. on Testing Computer Software, May 22-25, Washington, DC. Sponsors: DPMA, ACM. Contact Genevieve Houston-Ludlam, Frontier Technologies, 2444 Solomons Island Rd., Suite 205, Annapolis, MD 21401, phone (301) 266-8244.

 **10th Tunisian French Seminar of Computer Science: The Role of Languages in Programming, May 23-25**, Tunis, Tunisia. Cosponsor: Tunisian Information Processing

Society. Contact Abdelfettah Belghith, Dept. d'Informatique, Faculté des Sciences de Tunis, 1002 Belvedere Tunisia; or Ali Mili, Faculty of Sciences, Univ. of Tunis II, Campus Universitaire, 1002 Belvedere, Tunisia.

SIGMetrics 89 and Performance 89, May 23-26, Berkeley, Calif. Sponsors: ACM, IFIP. Contact Luis-Felipe Cabrera, IBM Almaden Research Center, Mail Code K52/803, San Jose, CA 95120-6099, phone (408) 927-1838.

ICCI 89, Int'l Conf. on Computing and Information, May 23-27, Toronto. Contact Waldemar W. Koczkodaj, Laurentian Univ., CoSc, Sudbury, Ont., Canada P3E 2C6, phone (705) 675-1151.

Workshop on New Directions in Computer Chess, May 28-June 1, Edmonton, Alta., Canada. Sponsors: Int'l Computer Chess Assoc., Canadian Information Processing Society. Contact Tony Marsland, Computing Science Dept., Univ. of Alberta, Edmonton, Alta., Canada T6G 2H1.

 **16th Int'l Symp. on Computer Architecture, May 28-June 1**, Jerusalem, Israel. Cosponsor: ACM. Contact M. Yoeli, Computer Science Dept., Technion City, Haifa 32000, Israel, phone 972 (4) 294-314.

 **19th Int'l Symp. on Multiple-Valued Logic, May 29-31**, Guangzhou, China. Cosponsors: Chongqing Univ. et al. Contact David M. Miller, Computer Science Dept., Univ. of Victoria, B.C., Canada, V8W 2Y2, phone (604) 721-7220.

CIPS National Congress 89, May 29-June 2, Edmonton, Alta., Canada. Sponsor: CIPS. Contact Congress 89, PO Box 1277, Main Post Office, Edmonton, Alta., Canada T5J 2M8, phone (403) 488-1879.

 **Int'l Conf. on Systolic Arrays, May 31-June 2**, Killarney, Ireland. Cosponsor: Queen's Univ. of Belfast. Contact Earl Swartzlander, TRW, 1 Space Park, Redondo Beach, CA 90278, phone (213) 535-4321.

June 1989

IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing, June 1-2, Victoria, B.C., Canada. Sponsor: IEEE Victoria Section, Univ. of Victoria. Contact Warren D. Little, Dept. of Electrical and Computer Engineering, Univ. of Victoria, PO Box 1700, Victoria, B.C., Canada V8W 2Y2, phone (604) 721-8686.

ICGA 89, Third Int'l Conf. on Genetic Algorithms, June 4-7, Washington, DC. Contact J. David Schaffer, Philips Labs, 345 Scarborough Rd., Briarcliff Manor, NY.

 **CVPR 89, Conf. on Computer Vision and Pattern Recognition, June 4-8**, San Diego, Calif. Contact Rama Chellappa, PHE324, Electrical Engineering-Systems

Dept., Univ. of Southern California, University Park, MC-0272, CA 90089, phone (213) 743-8559.

Fourth Israel Conf. on Computer Systems and Software Engineering, June 5-6, Tel Aviv. Cosponsors: Israel Chapter of IEEE Computer Society, Information Processing Assoc. of Israel. Contact S. Koenig, Ortra Ltd., PO Box 50432, Tel Aviv 61500, Israel, phone 972 (3) 664-825.

Fourth Symp. on Logic in Computer Science, June 5-8, Pacific Grove, Calif. Cosponsor: ACM. Contact Albert Meyer, Computer Science Lab, MIT, 545 Technology Square, Cambridge, MA 02139.

Ninth Int'l Conf. on Distributed Computing Systems, June 5-9, Newport Beach, Calif. Contact Kane Kim, Computer Engineering Program, Electrical Engineering Dept., Univ. of California, Irvine, CA 92717, phone (714) 856-5542.

Working Conf. on Computer-Aided Design Systems Using Artificial Intelligence Techniques, June 6-7, Tokyo. Cosponsors: IFIP, IPSJ. Contact Akihiko Yamada, NEC, 4-14-22 Shibaura, Minato-ku, Tokyo 108, Japan; or Kozo Kinoshita, Faculty of Integrated Arts and Sciences, Hiroshima Univ., 1-1-89 Sendamachi, Naka-ku, Hiroshima 730, Japan, phone 81 (82) 249-9150.

Third Int'l Workshop on Wafer Scale Integration, June 6-8, Como, Italy. Sponsor: IFIP. Contact Mariagiovanna Sami, Dip. di Elettronica, Politecnico di Milano, Piazza Leonardo da Vinci 32, I-20133 Milan, Italy, phone 39 (2) 23-99-35-16.

Second Int'l Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, June 6-9, Tullahoma, Tenn. Cosponsors: ACM, Univ. of Tennessee Space Inst. Contact Moonis Ali, Univ. of Tennessee Space Inst., Tullahoma, TN 37388, phone (615) 455-0631.

Ninth Int'l Symp. on Protocol Specification, Testing, and Verification, June 6-9, Enschede, The Netherlands. Sponsor: IFIP. Contact ICSC, Univ. of Twente, PO Box 217, 7500 AE Enschede, The Netherlands.

Eighth Biennial University/Government/Industry Microelectronics Symp., June 12-14, Westborough, Mass. Sponsors: Int'l Society for Hybrid Microelectronics, IEEE. Contact Richard B. Gold, Massachusetts Microelectronics Center, 75 North Dr., Westborough, MA 01581.

Workshop on Automatic Verification Methods for Finite State Systems, June 12-14, Grenoble, France. Sponsor: C-cube, the French National Project on Parallelism. Contact Edmund M. Clarke, Jr., Carnegie Mellon Univ., Computer Science Dept., Pittsburgh, PA 15213-3890; A. Pnueli, Weizmann Inst., Rehovot, Israel; or J. Sifakis, LGI-IMAG, BP 53X, 38041 Grenoble Cedex, France.

ICAIL 89, Second Int'l Conf. on Artificial Intelligence and Law, June 13-16, Vancouver, Canada. Contact Robert T. Franson or Joseph C. Smith, Faculty of Law, Univ. of British Columbia, Vancouver, B.C., Canada, phone (604) 228-2323.

Int'l Conf. on Software Engineering and Knowledge Engineering, June 15-17, Chicago. Contact Shi-Kuo Chang, Computer Science Dept., Univ. of Pittsburgh, 322 Alumni Hall, Pittsburgh, PA 15260, phone (412) 624-8490.

First Symp. on Parallel Algorithms and Architectures, June 18-21, Santa Fe, N.M. Sponsor: ACM. Contact Tom Leighton, Math Dept. and Computer Science Lab, MIT, Cambridge, MA 02139.

Int'l Workshop on Hardware Fault Tolerance in Multiprocessors, June 19-20, Urbana, Ill. Contact Prith Banerjee, Coordinated Science Lab, Univ. of Illinois, 1101 W. Springfield Ave., Urbana, IL 61801, phone (217) 333-6564.

CHDL 89, Ninth Int'l Symp. on Computer Hardware Description Languages and Applications, June 19-21, Washington, DC. Cosponsors: IFIP, ACM. Contact John A. Darringer, IBM T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598, phone (914) 945-1018.

Sixth Int'l Workshop on Database Machines, June 19-21, Deauville, France. Sponsors: INRIA, AFCET. Contact Haran Boral, MCC, 3500 W. Balcones Center Dr., Austin, TX 78759 (in the US); or Pascal Faudemay, Masi Labo., Univ. of Paris 6, 4 Place Jussieu, 75252 Paris, Cedex 05, France (outside the US).

ICNN 89, Int'l Conf. on Neural Networks, June 19-22, Washington, DC. Sponsor: IEEE. Contact Nomi Feldman, 3770 Tansy St., San Diego, CA 92121, phone (619) 453-6222.

Fourth Structure in Complexity Theory Conf., June 19-22, Eugene, Ore. Cosponsors: Univ. of Oregon, ACM. Contact Stephen R. Mahaney, Rm. 2C-454, AT&T Bell Labs, 600 Mountain Ave., Murray Hill, NJ 07974.

Graphics Interface 89, June 19-23, London, Ont., Canada. Sponsor: Canadian Man-Computer Communications Society. Contact Irene Gargantin, Computer Science Dept., Univ. of Western Ontario, London, Ont., Canada N6A 5B7, phone (519) 661-3563.

Vision Interface 89, June 19-23, London, Ont., Canada. Sponsor: Canadian Image Processing and Pattern Recognition Society. Contact Irene Gargantin, Computer Science Dept., Univ. of Western Ontario, London, Ont., Canada N6A 5B7, phone (519) 661-3563.

NECC 89, 10th National Educational Computing Conf., June 20-22, Boston. Cosponsor: Int'l Council for Computers in Education. Contact Michael C. Mulder,

Applied Research, Univ. of Portland, 5000 N. Willamette Blvd., Portland, OR 97203, phone (503) 283-7433; or NECC 89, ICCE, Univ. of Oregon, 1787 Agate St., Eugene, OR 97403-9905.

3rd Electronics Materials and Process Conf., June 20-22, Los Angeles. Sponsor: Society for the Advancement of Material and Process Engineering. Contact Marge Smith, SAMPE, PO Box 2459, 843 W. Glentana, Covina, CA 91722, phone (818) 331-0616.

FTCS 19, Int'l Fault-Tolerant Computing Symp., June 21-23, Chicago. Contact S.M. Reddy, FTCS 19, Electrical and Computer Engineering Dept., Univ. of Iowa, Iowa City, IA 52242, phone (319) 335-5196; or Ravi K. Iyer, Coordinated Science Lab, Univ. of Illinois, 1101 W. Springfield Ave., Urbana, IL 61801, phone (217) 333-9732.

Third Int'l Conf. on Foundations of Data Organization and Algorithms, June 21-23, Paris. Sponsor: INRIA. Contact Witold Litwin, INRIA Rocquencourt, c/o Public Relations Dept., Domaine de Voluceau, 78153 Le Chesnay Cedex, France, phone 33 (1) 39-63-56-00.

SIGPlan 89, Conf. on Programming Language Design and Implementation, June 21-23, Portland, Ore. Sponsor: ACM. Contact Bruce Knobe, Prime Computer, Inc., 500 Old Connecticut Path, Framingham, MA 01701, phone (508) 879-2960.

Second Symp. on Computer-Based Medical Systems, June 25-27, Minneapolis, Minn. Cosponsors: IEEE, Univ. of Minnesota. Contact John M. Long, 2829 University Ave. SE, #408, Minneapolis, MN 55414, phone (612) 627-4850; or Bart Galle, Continuing Medical Education, Univ. of Minnesota, Box 202 UMHC, 420 Delaware St. SE, Minneapolis, MN 55455, phone (612) 626-5525.

CAR 89, Computer-Assisted Radiology Conf., June 25-28, Berlin. Cosponsor: AMK Berlin. Contact Michael L. Rhodes, MPDI, 2730 Pacific Coast Hwy., Torrance, CA 90505, phone (213) 539-5944.

DAC 89, 26th Design Automation Conf., June 25-29, Las Vegas. Cosponsor: ACM. Contact Michael J. Lorenzetti, MCNC, PO Box 12889, Research Triangle Park, NC 27709-2889; or Pat Pistilli, MP Associates, 7490 Clubhouse Rd., Suite 102, Boulder, CO 80301, phone (303) 530-4333.

Hot Chips Symp., June 26-27, Palo Alto, Calif. Sponsor: IEEE Computer Society Technical Committee on Microprocessors and Microcomputers. Contact Jack Grimes, Intel Corp., SC4-40, 3065 Bowers Ave., Santa Clara, CA 95051; or Dave Ditzel, Sun Microsystems, A7-41, 2550 Garcia Ave., Mountain View, CA 94043.

Int'l Assoc. of Knowledge Engineers Conf., June 26-28, College Park, Md. Cosponsors: IAKE, Univ. of Maryland. Contact Fred Whiting, IAKE Conf., Georgetown PO Box 25461, Washington, DC 20007, phone (301) 231-7826.

Int'l Conf. on Expert Planning Systems, June 27-29, Brighton, UK. Sponsor: Institution of Electrical Engineers. Contact Conf. Services, IEE, Savoy Pl., London WC2R 0BL, UK, phone 44 (1) 240-1871, ext. 222.

IFCS 89, Second Conf. of the Int'l Federation of Classification Societies, June 27-30, Charlottesville, Va. Contact IFCS 89, Mathematics Dept., Univ. of Virginia, Charlottesville, VA 22903, phone (804) 924-4919.

July 1989

SETSS 90, Seventh Int'l Conf. on Software Engineering for Telecommunications Switching Systems, July 3-6, Bournemouth, England. Sponsor: Institution of Electrical Engineering. Contact Conf. Services, IEE, Savoy Pl., London WC2R 0BL, UK, phone 44 (01) 24-01-871.

EKAU 89, Third European Knowledge Acquisition for Knowledge-Based Systems Workshop, July 3-7, Paris. Contact John Boose, Advanced Technology Center, Boeing Computer Services, 7L-64, PO Box 24346, Seattle, WA 98124, phone (206) 865-3253; or Brian Gaines, Dept. of Computer Science, Univ. of Calgary, 2500 University Dr. NW, Calgary, Alta., Canada T2N 1N4, phone (403) 220-5901.

ICALP 89, 16th Colloquium on Automata, Languages, and Programming, July 11-15, Stresa, Italy. Sponsor: European Assoc. of Theoretical Computer Science. Contact Ronchi Della Rocca, Dip. di Informatica, corso Svizzera 185, 10149 Torino, Italy, phone 39 (11) 75-56-77.

Eighth Australia Conf. on Microelectronics: July 12-14, Brisbane, Australia. Cosponsors: IEEE Queensland Section, Univ. of Queensland. Contact Microelectronics 89, Institution of Engineers, Australia, 11 National Circuit, Barton ACT 2600, Australia, phone 61 (62) 706-549.

Symp. on Design and Implementation of Large Spatial Databases, July 17-18, Santa Barbara, Calif. Contact Oliver Gunther, Computer Science Dept., Univ. of California, Santa Barbara, CA 93106, phone (805) 961-3236.

CASE 89, Third Int'l Workshop on Computer-Aided Software Engineering, July 17-21, London. Cosponsors: Imperial College of London et al. Contact Elliot Chikofsky, Index Technology Corp., 1 Main St., Cambridge, MA 02142 (in North America); or John Jenkins, School of Management, Imperial College, London SW7 2PG, UK (in other regions).

CALL FOR PAPERS

Advances in Parallel and Distributed Processing plans a special issue on applications. Publisher: Ablex Publishing. Submit abstract to Harry Tyrer, Electrical and Computer Engineering Dept., Univ. of Missouri at Columbia, Columbia, MO 65211, phone (314) 882-6489.

Hot Chips Symp.: June 26-27, 1989, Palo Alto, Calif. Sponsor: IEEE Computer Society Technical Committee on Microprocessors and Microcomputers. Submit proposal for presentation to Jack Grimes, Intel Corp., SC4-40, 3065 Bowers Ave., Santa Clara, CA 95051; or Dave Ditzel, Sun Microsystems, A7-41, 2550 Garcia Ave., Mountain View, CA 94043.

J. Software Maintenance: Research and Practice will begin publication this year. Publisher: John Wiley and Sons. Submit paper to Keith H. Bennett, Center for Software Maintenance, South Road, Univ. of Durham, Durham, DH1 3LE, UK, phone (091) 374-2632; or Mel A. Colter, Colter Enterprises, 19520 Indian Summer Lane, Monument, CO 80132, phone (719) 481-4040.

Int'l J. Computer-Aided VLSI Design plans a special issue on placement and routing. Publisher: Ablex Publishing. Submit paper by **Feb. 28, 1989**, to Mehdi Zargham, Computer Science Dept., Southern Illinois Univ., Carbondale, IL 62901-4511, phone (618) 536-2327.

IEEE Software: November 1989. Articles are sought for a special edition on reverse engineering. Submit manuscript by **Mar. 1, 1989**, to Elliot Chikofsky, Index Technology Corp., 1 Main St., Cambridge, MA 02142, phone (617) 494-8200.

IEEE Trans. Computers: December 1989. A special issue is planned on computer systems performance. Submit manuscript by **Mar. 1, 1989**, to Edward D. Lazowska, Computer Science Dept. FR-35, Univ. of Washington, Seattle, WA 98195, phone (206) 543-4755.

Int'l J. Approximate Reasoning plans a special issue on belief functions and belief maintenance in artificial intelligence. Submit paper by **Mar. 1, 1989**, to Prakash P. Shenoy, School of Business, Univ. of Kansas, Summerfield Hall, Lawrence, KS 66045-2003, phone (913) 864-7551; or Gautam Biswas, Computer Science Dept., Box

1688, Station B, Vanderbilt Univ., Nashville, TN 37235, phone (615) 343-6204.

Workshop on New Directions in Computer Chess: May 28-June 1, 1989, Edmonton, Alta., Canada. Sponsors: Int'l Computer Chess Assoc., Canadian Information Processing Society. Submit paper or abstract by **Mar. 1, 1989**, to Tony Marsland, Computing Science Dept., Univ. of Alberta, Edmonton, Alta., Canada T6G 2H1.

Fifth Int'l Conf. on Image Analysis and Processing: Sept. 20-22, 1989, Positano, Italy. Sponsors: Int'l Assoc. for Pattern Recognition et al. Submit abstract by **Mar. 1, 1989**, to Gabriella Sanniti di Baja, c/o Istituto di Cibernetica, C.N.R., I-80072 Arco Felice, Naples, Italy, phone 39 (81) 867-1255.

32nd Midwest Symp. on Circuits and Systems: Aug. 14-15, 1989, Urbana, Ill. Sponsors: Univ. of Illinois at Urbana-Champaign, IEEE. Submit paper by **Mar. 1, 1989**, to Sung Mo Kang, Univ. of Illinois, 1101 W. Springfield Ave., Urbana, IL 61801-3082, phone (217) 244-0577.

89 VLSI Education Conf. and Exposition: July 19-21, 1989, Santa Clara, Calif. Submit paper by **Mar. 1, 1989**, to Randy Katz, Conf. Management Services, 5 Cleland Pl., Menlo Park, CA 94025, phone (415) 329-0510.

28th Technical Symp.: Aug. 14, 1989, Gaithersburg, Md. Sponsors: Washington, DC Chapter of ACM, NIST. Submit paper by **Mar. 2, 1989**, to Milton S. Hess, American Management Systems, 1525 Wilson Blvd., Arlington, VA 22209.

Manufacturing Int'l 90 Conf.: Mar. 25-28, 1990, Atlanta. Sponsor: American Society of Mechanical Engineers. Submit abstract by **Mar. 3, 1989**, to Salah E. Elmaghraby, North Carolina State Univ., c/o ASME, 345 E. 47th St., New York, NY 10017.

First Int'l Conf. on Artificial Neural Networks: Oct. 17-18, 1989, London. Sponsor: Institution of Electrical Engineers. Submit synopsis by **Mar. 3, 1989**, to Conf. Services, IEE, Savoy Pl., London WC2R 0BL, UK, phone 44 (1) 24-01-871.

Beijing Int'l Symp. for Young Computer Professionals: Aug. 21-23, 1989, Beijing. Sponsors: China Assoc. for Science and Technology, Chinese Computer Federation. Submit paper by **Mar. 15, 1989**, to Siping

Zhang, Inst. of Computing Technology, Academia Sinica, PO Box 2704, Beijing, China 100080.

GOMAC 89, Government Microcircuit Applications Conf.: Nov. 7-9, 1989, Orlando, Fla. Sponsors: US Dept. of Energy et al. Submit summary by **Mar. 17, 1989**, to Jay Morreale, G-89, Palisades Inst. for Research Services, 201 Varick St., Rm. 1140, New York, NY 10014.

11th Electrical Overstress/Electrostatic Discharge Symp.: Sept. 26-28, 1989, New Orleans. Sponsors: IIT Research Inst., EOS/ESD Assoc. Submit paper by **Mar. 17, 1989**, to Bob Rountree, Texas Instruments, 12201 Southwest Frwy., MS 631, Houston, TX 77001, phone (713) 274-4077.

Crypto 89: Aug. 20-24, 1989, Santa Barbara, Calif. Sponsor: Int'l Assoc. for Cryptologic Research. Submit abstract by **Mar. 17, 1989**, to Gilles Brassard, IRO Dept., Univ. of Montreal, CP 6128, Succursale "A", Montreal, Que., Canada H3C 3J7, phone (514) 343-6807.

Int'l J. Computer Aided VLSI Design plans a special issue of neural networks. Publisher: Ablex Publishing. Submit paper by **Mar. 28, 1989**, to Omid Omidvar, Computer Science Dept., Univ. of the District of Columbia, 4200 Connecticut Ave. NW, Washington, DC 20008, phone (202) 282-7345.

First Int'l Workshop on Human and Machine Cognition: May 11-13, Santa Rosa Island, Fla. Submit paper by **Mar. 30, 1989**, to Keh Ford, Computer Science Div., Univ. of West Florida, Pensacola, FL 32514, phone (904) 474-2551.

Int'l Assoc. of Knowledge Engineers Conf.: June 26-28, 1989, College Park, Md. Cosponsors: IAKE, Univ. of Maryland. Submit paper by **Mar. 31, 1989**, to Michael Teague, IAKE, Georgetown, PO Box 25461, Washington, DC 20007.

Eighth Symp. on Reliable Distributed Systems: Oct. 10-12, 1989, Seattle, Wash. Submit paper by **Apr. 1, 1989**, to Raif Yanney, TRW, 1 Space Park, DH2-2328, Redondo Beach, CA 90278, phone (213) 812-6033.

Symp. on Design and Implementation of Large Spatial Databases: July 17-18, 1989, Santa Barbara, Calif. Submit paper by **Apr. 1, 1989**, to Oliver Gunther, Computer Science Dept., Univ. of California, Santa Barbara, CA 93106, phone (805) 961-3236.

Seventh Pacific Northwest Software Quality Conf.: Sept. 11-12, 1989, Portland, Ore. Submit abstract by **Apr. 3, 1989**, to Dick Hamlet, Computer Science Dept., Portland State Univ., PO Box 751, Portland, OR 97207-0751.

CASE 89, Third Int'l Workshop on Computer-Aided Software Engineering: July 17-21, 1989, London. Cosponsors: Imperial College of London et al. Submit

position paper by **Apr. 7, 1989**, to Elliot Chikofsky, Index Technology Corp., 1 Main St., Cambridge, MA 02142 (for North America); or John Jenkins, School of Management, Imperial College, London SW7 2PG, UK (for other regions).

Int'l Symp. on Computer Architecture and Digital Signal Processing: Oct. 11-14, 1989, Hong Kong. Sponsor: IEEE. Submit paper by **Apr. 7, 1989**, to W.C. Siu, Electronic Engineering Dept., Hong Kong Polytechnic, Hong Kong, phone (852) 3-638344, ext. 496.

Second IEEE Workshop on Workstation Operating Systems: Sept. 27-29, 1989, Pacific Grove, Calif. Submit position statement by **Apr. 15, 1989**, to Luis-Felipe Cabrera, Mail Code K52/803, IBM Almaden Research Center, 650 Harry Rd., San Jose, CA 95120-6099.

Eighth Int'l Conf. on Entity-Relationship Approach: Oct. 18-20, 1989, Toronto. Sponsor: ER Inst. Submit paper by **Apr. 15, 1989**, to Frederick H. Lochovsky, Computer Science Dept., Univ. of Toronto, Stanford Fleming Bldg., 10 King's College Circle, Toronto, Ontario M5S 1A4, Canada, phone (416) 978-7441.

Micro 22, 22nd Workshop on Microprogramming and Microarchitecture: Aug. 14-16, 1989, Dublin, Ireland. Submit paper by **Apr. 15, 1989**, to Gearold R. Johnson, Center for Computer Assisted Engineering, Colorado State Univ., Fort Collins, CO 80523, phone (303) 491-5543.

13th Western Educational Computing Conf.: Nov. 16-17, 1989, Burlingame, Calif. Sponsor: California Educational Computing Consortium. Submit paper by **Apr. 20, 1989**, to Oliver Seely, Jr., Chemistry Dept., California State Univ. at Dominguez Hills, 1000 E. Victoria St., Carson, CA 90747.

Third Int'l Workshop on Distributed Algorithms: Sept. 26-28, 1989, Nice, France. Submit paper by **Apr. 25, 1989**, to Jean-Claude Bermond, LRI, Bat 490, Universite Paris-Sud, 91405 Orsay, France; or Michel Raynal, IRISA, Campus de Beaulieu, 35042 Rennes, France.

ISCS 4, Fourth Int'l Symp. on Computer and Information Sciences: Oct. 30-Nov. 1, 1989, Cesme, Turkey. Submit paper by **Apr. 25, 1989**, to Asuman Dogac, Computer Engineering Dept., Middle East Technical Univ., 06531 Ankara, Turkey.

Int'l J. Computer Aided VLSI Design plans a special issue on design simulation and implementation. Publisher: Ablex Publishing Corp. Submit paper by **Apr. 30, 1989**, to Harry W. Tyrer, Electrical and Computer Engineering Dept., University of Missouri, Columbia, MO 65211, phone (314) 882-6489.

Supercomputing 89: Nov. 13-17, 1989, Reno, Nev. Cosponsor: ACM. Submit paper by **May 1, 1989**, to Gary Johnson, San Diego Supercomputer Center, PO Box 85608, San Diego, CA 92138, phone (619) 534-5181.

Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop: Oct. 1-6, 1989, Banff, Canada. Sponsor: American Assoc. for Artificial Intelligence. Submit draft paper by **May 1, 1989**, John H. Boose, Advanced Technology Center, Boeing Computer Services, 7L-64, PO Box 24346, Seattle, WA 98124, phone (206) 865-3253.

J. Microcomputer Applications: January 1990. A special issue is planned on transporter applications. Submit paper by **May 1, 1989**, to F.J. Rammig, Paderborn Univ., FB 17, Warburger Str. 100, 4790 Paderborn, West Germany, phone 49 (05251) 60-20-69.

Int'l Conf. on Expert Planning Systems: June 27-29, 1989, Brighton, UK. Sponsor: Institution of Electrical Engineers. Submit paper by **May 5, 1989**, to Conf. Services, IEE, Savoy Pl., London WC2R 0BL, UK, phone 44 (1) 240-1871, ext. 222.

IEEE Software: March, 1990. A special issue is planned on software development metrics. Submit paper by **May 15, 1989**, to Peter Dyson, Software Productivity Solutions, 122 N. Fourth Ave., Indialantic, FL 32903, phone (407) 984-3370.

14th Conf. on Local Computer Networks: Oct. 10-12, 1989, Minneapolis. Submit paper by **May 15, 1989**, to Larry Green, Protocol Engines, 1421 State St., Santa Barbara, CA 93101, phone (805) 965-0825.

Ninth Conf. on Foundations of Software Technology and Theoretical Computer Science: Dec. 19-21, 1989, Bangalore, India. Submit paper by **May 15, 1989**, to C.E. Veni Madhavan, Tata Research Development and Design Center, 1 Mangaldas Rd., Pune 411001, India, phone (212) 662-453.

Fifth Int'l Congress on Advances in Non-Impact Printing Technologies: Nov. 12-17, 1989, San Diego, Calif. Sponsor: Society for Imaging Science and Technology. Submit abstract by **May 22, 1989**, to John Moore, Tektronix, PO Box 500, MS 50-321, Beaverton, OR 97077, phone (503) 627-5067.

Second Int'l Symp. on Artificial Intelligence: Oct. 23-27, 1989, Monterrey, Mexico. Submit abstract and paper by **May 31, 1989**, to Francisco J. Cantu, Inst. Tecnológico de Monterrey, ITESM Suc. Correos "J", Monterrey, N.L., Mexico 64849, phone 52 (83) 58-20-00.

IEEE Workshop on Tools for Artificial Intelligence: Oct. 23-25, 1989, Fairfax, Va. Submit paper or summary by **June 15, 1989**, to N.G. Bourbakis, George Mason Univ., ECE Dept., Machine Vision Lab, Fairfax, VA 22030.

Fourth Int'l Workshop on High-Level Synthesis: Oct. 15-18, 1989, Kennebunkport, Maine. Cosponsor: ACM. Submit extended abstract by **June 16, 1989**, to Raul Camposano, IBM Research Div., T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598, phone (914) 945-3971.

CAREER OPPORTUNITIES

RATES: \$12.00 per line, \$120 minimum charge (up to ten lines). Average five typeset words per line, eight lines per column inch. Add \$10 for box number. Send copy at least one month prior to publication to: **Heidi Rex or Marian Tibayan, Classified Advertising, COMPUTER Magazine, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 821-8380.**

TRINITY COLLEGE

The Department of Engineering and Computer Science invites applications from outstanding candidates for a tenure-track position at the Assistant Professor-level commencing September, 1989, in the areas of FLUID MECHANICS/HEAT TRANSFER or ROBOTICS/CONTROLS. Experimental background highly desirable. The position involves graduate and undergraduate instruction and research, and a doctoral degree is a prerequisite. We are interested in receiving applications from qualified women and minorities. Trinity College is an equal opportunity/affirmative action employer. Please send resume to Professor Joseph D. Bronzino, Chairman, ECS Department, Trinity College, Hartford, CT 06106. Applications will be accepted until February 15, 1989.

CALIFORNIA STATE UNIVERSITY San Bernardino

POSITION FOR ASSISTANT OR ASSOCIATE PROFESSOR (tenure track) of **COMPUTER SCIENCE**. Salary range \$33,660-\$52,068 dependent on qualifications and experience. Moving expenses. Available January, April or September 1989.

Duties include teaching, advising, curriculum development, research, and community interaction. Teaching load equated to 12 hrs/wk of lecture and lab. Ph.D. in CSci is required. Facilities include CDC, Prime, and DEC computers, many PCs, microprocessors, terminals and several LANs.

The University is located in one of the fastest growing service areas in the USA with a current student population of about 9000 and 300 computer science majors. The area is noted for its warm climate and is within a 1-2 hr drive of Palm Springs, mountain ski resorts, beaches, and metropolitan Los Angeles. Housing costs average 20% lower than the LA area. Applicants should submit letter of application and vitae. Women and minorities are encouraged to apply. Position open until filled but no later than May 15. Send materials to: Dr. Richard J. Botting, Chair, Computer Science Dept., California State University, 5500 University Parkway, San Bernardino, CA 92407, PAAAAAR@CALSTATE.BITNET

An Equal Opportunity/Affirmative Action, Section 504, Title IX Employer.

GEORGIA INSTITUTE OF TECHNOLOGY School of Information and Computer Science

The School of Information and Computer Science invites applications for faculty positions at all levels. Applicants should have a commitment to teaching and a record of outstanding research accomplishments. Applicants who expect to receive a Ph.D. degree by Fall 1989 and who show high potential for research as well as a commitment to teaching are also invited.

The School seeks applicants to strengthen its capabilities in all areas of computer science. Very competitive salaries are offered.

The School has 30 faculty members and anticipates further faculty growth. Its educational activities include an undergraduate program accredited by the Computing Sciences Accreditation Board, Inc., a Masters program with 150 students and a Ph.D. program with over 70 students. Well equipped laboratories support research and education. High-speed local area networks interconnect all major campus laboratories and provide access to national networks.

The School is in the second year of a five year Coordinated Experimental Research grant from the National Science Foundation. This grant is funding acquisition of hardware and development of software to support experimental work in parallel and distributed computing.

Georgia Tech is located in Atlanta, which experiences a mild sunbelt climate. It is the center of commerce in the Southeast, offering a diverse economy and good employment opportunities in all professional areas. Atlanta offers good cultural and recreational opportunities, extremely attractive residential neighborhoods, and affordable housing.

Candidates should send complete resumes and names of at least three references to: Chairman, Faculty Search Committee; School of Information and Computer Science; Georgia Institute of Technology; Atlanta, Georgia 30332.

Georgia Tech is an equal opportunity employer.

PRINCETON UNIVERSITY

The Department of Electrical Engineering invites applications for a full time, tenure-track faculty position. The discipline of particular interest is Computer Engineering with a specialization in an area such as: CAD for integrated circuits and systems, fault tolerance and computer architecture, VLSI, automated synthesis of digital systems. Please send a complete resume, a description of research and teaching interests and names of three references to Professor Stuart Schwartz, Chairman, Dept. of EE, Princeton University, Princeton, NJ 08544. An Affirmative Action/Equal Opportunity Employer.

UNIVERSITY OF HOUSTON

Applications are invited for tenure track faculty positions in the Department of Computer Science starting September 1989. Areas of special interest include but not limited to artificial intelligence, computer architecture, computer graphics, computer networks, operating systems, programming languages and software engineering. Rank and salary are open and competitive. The Department is interested in expanding its research program and particularly welcome applicants for senior positions. Applicants should have a Ph.D. in Computer Science or a closely related area, and a strong commitment to research and teaching. Candidates for senior positions should also have a distinguished research record. The Department offers Ph.D., M.S., and B.S. in Computer Science. Departmental research facilities include a network of SUN Workstations, VAX 11/780 and VAX 11/730's, a network of AT+T 3B20 and 3B2's and access to other computing facilities in the University Computer Center as well as supercomputers via remote access terminals. Send resume and names of professional references to Dr. Willis King, Chairman, Department of Computer Science, University of Houston, Houston, Texas 77204-3475. An Equal Opportunity/Affirmative Action Employer.

THE UNIVERSITY OF MICHIGAN Computer Science and Engineering Division

The Department of Electrical Engineering and Computer Science at The University of Michigan invites applications for tenure-track positions in its Computer Science and Engineering Division. Located in a new building on the North Campus and with a strong commitment from the University, the Division is entering an exciting period of further expansion.

The University of Michigan has a long history in software development, and is interested in expanding its coverage of the field particularly at mid-career or senior levels. Possible appointments are available primarily in the areas of operating systems, software engineering and networks. In theoretical computer science we seek a junior faculty member with interests compatible with our current efforts in logic or parallel algorithms. All candidates who apply should have an interest in teaching and a strong research orientation.

Send your resume and the names of at least three references to Professor Keki B. Irani, Associate Chairman, Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109-2122. The University of Michigan is an Equal Opportunity/Affirmative Action Employer.

UNIVERSITY OF WASHINGTON

The University of Washington seeks candidates for tenure-track faculty appointments in computer science/engineering starting in the 1989-90 academic year. Applications from outstanding individuals in all areas of computer science/engineering will be considered.

A moderate teaching load allows time for quality research and close involvement with students. We expect applicants to have a strong commitment to both research and teaching, and an outstanding record of research for their level. Any appointment should bring significant new research strength to the University.

Interested applicants should send a letter of application, a resume, and the names of four references to: Computer Science/Engineering Faculty Recruiting Committee, FR-35, University of Washington, Seattle, WA 98195.

The University of Washington is an Affirmative Action/Equal Opportunity Employer. The Ph.D. is required for these positions.

UNIVERSITY OF CALIFORNIA AT IRVINE

Faculty Positions in Computer Science

The Department of Information and Computer Science (ICS) is actively recruiting new faculty members. We have energetic research groups in the areas of architecture and operating systems, artificial intelligence and machine learning, software engineering and programming environments, social and managerial analysis of computing, and data structures and algorithms. We are currently building on these areas of existing strength.

We are looking for energetic, congenial candidates who would thrive in a serious but friendly research setting, and who would like to join us in exploring the nature of computing, broadly defined. Exceptionally distinguished candidates for senior positions are especially encouraged to apply.

The ICS Department is an independent academic unit reporting to the Executive Vice Chancellor. ICS maintains an emphasis on core computer science as well as effective interdisciplinary ties to colleagues in Neurobiology, Cognitive Science, Management, Engineering, and other areas. The department currently has 25 full-time faculty positions and 85 Ph.D. students, with major support from the administration to expand and to strengthen the research environment. Annual research contracts from DARPA, NSF, ONR, and other agencies currently total over \$3 million. In 1986 the Department was awarded a Coordinated Experimental Research (CER) grant from the National Science Foundation. This support is fostering the creation of a Laboratory for Software Research, in which major studies of the development and evaluation of software technology can be undertaken. It is also being used to strengthen the base of machines supporting other software research. Department equipment includes over 100 workstations, primarily Sun-3's. Two large multiprocessor Sequents are available, as well as approximately 75

Macintosh Plus's and II's. Several laser printers are available for high-quality output. All machines are tied together with networks, which are gatewayed to the campus network, and from there, to regional, national, and international networks (Darpa Internet, CSnet, Bitnet, etc.). In addition, department members have access to campus-wide computing resources as well as regional supercomputer access.

UCI is located in Orange County, three miles from the Pacific Ocean near Newport Beach, and approximately halfway between Los Angeles and San Diego. The campus is situated in the heart of a national center of high-technology enterprise, and is experiencing substantial growth with exciting opportunities. Salaries and benefits are competitive. Special housing assistance is available from the university, including newly built, for-sale housing within short walking distance from the Department.

Send resumes and names of four references to:

John L. King, Chair
Department of Information
and Computer Science
University of California
Irvine, CA 92717

Apply before March 1, 1989, to ensure consideration.

An Affirmative Action/Equal Opportunity Employer.

UNIVERSITY OF NORTH TEXAS

Department Chair Computer Sciences

Applications and nominations are invited for the position of Chair of the Department of Computer Sciences. The department offers B.S., M.S. and Ph.D. programs with approximately 500 undergraduate and 150 graduate majors. Current strengths among the 19 faculty include parallel and distributed computing, logic programming, artificial intelligence, expert systems, databases, languages, numerical computation, and operating systems. Substantial faculty growth is projected over the next five years. Excellent facilities support both research and instruction.

The University of North Texas is a comprehensive, graduate research university of approximately 24,000 students, offering an array of undergraduate, master's and doctoral programs. The university is located in Denton, at the center of one of the fastest growing counties in the nation, less than forty miles from both Dallas and Fort Worth. The Dallas-Fort Worth-Denton Metroplex abounds in cultural activities and high technology industries.

Substantial research accomplishments are essential; grant activity and administrative experience are desirable. Apply to Chair Search Committee, Department of Computer Sciences, Box 13886, University of North Texas, Denton, TX 76203. Applications should include a curriculum vitae and the names, addresses and telephone numbers of at least three references. Applications will be received until the position is filled.

UNT is an equal opportunity/affirmative action employer.

UNIVERSITY OF SOUTHERN CALIFORNIA

Chair, Computer Science

The Computer Science Department of the School of Engineering, University of Southern California (USC), seeks a distinguished computer scientist with the vision and skill to lead a strong department and further strengthen it. The School of Engineering, with 170 faculty and 4,300 students, ranks 6th in the nation in sponsored research expenditures. The Computer Science Department has a full time faculty of 22, and a student body of 125 Ph.D. students, 250 M.S. candidates, and 200 undergraduates. The departmental research budget currently is \$3M per year. Faculty research interests encompass the traditional areas of computer science, plus interdisciplinary, emerging fields such as robotics and neural computation. Computing research support is provided by a large network of modern workstations, and various supercomputers are available through the network. More than 100 SUN workstations are used for teaching. The department maintains close ties with the Electrical Engineering Department which has a strong Computer Engineering Group of 16 faculty, and with the Information Sciences Institute (ISI), an off-campus research facility of the School of Engineering. ISI's staff of 200 conducts research on a broad spectrum of computer science topics. Southern California has the highest industrial production in the nation. Opportunities for industry/university collaboration are excellent because much of the local industry is high-tech and computationally oriented.

Please send nominations and applications to:

Professor Aristides Requicha
Chair, Search Committee
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782
Telephone: (213) 743-3805
Net Mail: requicha@lipari.usc.edu

Applications should include a resume and a list of professional references. USC is an equal opportunity employer.

UNIVERSITY OF MARYLAND

UNIVERSITY COLLEGE

Faculty for Europe and Asia

Planning a sabbatical or leave of absence? The University of Maryland University College seeks excellent lecturers for undergraduate computer science, computer applications, and information systems management courses on U.S. military bases in Europe and in Asia and the Pacific. Renewable annual appointments begin August 1989. Minimum requirements include a master's degree in computer science or a related field, recent college teaching experience, and U.S. citizenship. Benefits include transportation and military base privileges. Frequent travel and the cost of schooling make these positions difficult for those with children. Send resume to Dr. Ralph E. Millis, Overseas Programs, The University of Maryland University College, College Park, MD 20742-1642. AA/EEO.

THE UNIVERSITY OF ARIZONA

The Department of Computer Science at the University of Arizona invites applications for faculty positions at all ranks to begin in August, 1989. Applicants must have a doctorate in Computer Science or a closely related field. Applicants for senior positions should have made substantial research contributions to the field, while applicants for junior positions should show promise of future excellence.

The Department of Computer Science at Arizona emphasizes excellence in research and teaching. There are currently 12 faculty members, with plans to expand over the next few years as the department institutes a selective undergraduate major. Research is currently conducted in a variety of areas including programming languages, software systems, parallel and distributed computing, logic programming, and theory of computation. Qualified individuals working in these areas—as well as other areas such as artificial intelligence, computer architecture, scientific computation, and performance analysis—are encouraged to apply.

The research program is supported by numerous grants to individual faculty as well as a department-wide NSF grant for Coordinated Experimental Research (CER). Computational facilities include a VAX 8650, dozens of Sun workstations, an Intel iPSC Hypercube, and an HP 9000 graphics workstation. Also available are high-resolution color terminals, microcomputers, laser printers, and a phototypesetter. A soon-to-expand instructional laboratory contains two VAX 11/785.

Send a complete resume and the names of at least three references to Richard D. Schlichting, Faculty Recruiting Committee Chairman, Department of Computer Science, The University of Arizona, Tucson, AZ 85721. Applications will be reviewed beginning January 15, 1989, but the positions will remain open until filled. The University of Arizona is an equal opportunity/affirmative action employer.

UNIVERSITY OF PITTSBURGH Department of Computer Science

The Department of Computer Science has entered a period of substantial growth and invites applications for four tenure track faculty positions to be filled by June, 1989 or as soon as possible. Although we are primarily seeking candidates at the assistant professor level, we will also consider candidates for appointment at more senior ranks, commensurate with demonstrated scholarly achievements. Responsibilities include research, supervision of graduate student research (Ph.D. and M.S.), and graduate and undergraduate teaching. Candidates should have a Ph.D. in computer science or in a closely related field and a strong interest in both research and teaching. All areas of specialization in computer science will be considered, with preference given to programming languages, software engineering, operating systems and artificial intelligence.

The Department currently has twenty-one full-time faculty members and supports strong graduate and undergraduate programs. Departmental resources include an excellent research library and numerous computing facilities including a network of SUN and Xerox 1100-series (Dandelion) workstations, a VAX 11/780 (under BSD UNIX), a variety of micro-computers, and several graphics systems. The research systems are accessible via the Department's Ethernet-compatible LAN. Convenient access is also provided to the extensive general computer facilities of the University as well as to other networks (e.g., ARPANET, CSNET). The Department operates the *Center for Parallel, Distributed and Intelligent Systems* (CPDIS) to provide an environment for innovative research in computer science. Since the University of Pittsburgh is a founding member of the Pittsburgh Supercomputing Center and an affiliate member of the Software Engineering Institute, the Department of Computer Science has access to the Cray XMP/48 of PSC and the software engineering expertise at SEI.

Please send your resume to: Dr. Mary Lou Soffa, Chair of Faculty Search, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260.

Pitt is an equal opportunity/affirmative action employer and especially encourages women and members of ethnic minorities to apply.

UNIVERSITY OF NEW ORLEANS Professorships in Computer Science

We invite applications for tenure-track positions in the Computer Science Department at the University of New Orleans. Ours is a rapidly-developing program with approximately 400 majors at the second-largest public university in Louisiana. We also offer graduate study through the Master's level in a cooperative program with the Department of Mathematics. Our Department numbers 11 full-time faculty with good prospects for continued expansion for several years. Our undergraduate program has been accredited by the Computing Sciences Accreditation Board, and was also singled out for excellence by the Louisiana Board of Regents in the past year. Only two universities in Louisiana have received both of these recognitions.

University computing facilities include a network of four VAX 8600's running VMS 4.5 and DECNET, and approximately 150 Z-100 (MS-DOS compatible) 10 MB microcomputers which serve as the principal workstations on campus. There is also a campus-wide Ethernet. Departmental facilities include a Gould 6040/Berkeley 4.3bsd Departmental computing system, a Symbolics 3640 LISP machine, and several Sun workstations, as well as a number of microcomputers and a PDP-11. We have now completed a purchase of a large number of advanced workstations to complement the existing facilities, including one for each faculty member. We are also a member of CSNET. Ada is used as the primary programming language in undergraduate teaching.

Candidates will be considered for both junior and senior positions. Candidates should have a doctorate in Computer Science or in a related field, and should be interested in a career where teaching excellence is rewarded and research encouraged. Our primary objective is for faculty in the areas of operating systems, database management systems, and software engineering; candidates whose specialties are in other branches of computer science will only be considered under exceptional circumstances.

Please send your resume and three references before March 15, 1989 to Wayne Patterson, Chairman, Department of Computer Science, University of New Orleans, New Orleans, LA 70148. (Phone: 504-286-6594.) Inquiries may also be directed via Bit-Net to *rwpcs@uno* or via CSNet to *wpatters@uno*. Review of the applications will begin upon their receipt. Women and minorities are especially encouraged to apply. The University of New Orleans is an Equal Opportunity/Affirmative Action Employer.

UNIVERSITY OF WISCONSIN- MILWAUKEE Distinguished/Chaired Professor in Computer Science Department of Electrical Engineering and Computer Science

The Department of Electrical Engineering and Computer Science of the University of Wisconsin-Milwaukee invites nominations and applications for the position of a Distinguished/Chaired Professor in Computer Science. It is expected that candidates will have an outstanding reputation in research, a strong interest in teaching and a commitment to the development of Computer Science at the University. It is also desirable that the selected candidate would develop an interaction with the local industry. The main interests of the candidates should be in one of the following areas: Artificial Intelligence, Computer Architecture, Computer Systems, and Software Engineering. Salary and other benefits are competitive.

The Department offers undergraduate and graduate programs in Electrical Engineering as well as Computer Science. The Computer Science faculty strength is 13 and is expected to grow. The current areas of faculty research strengths include: Data Security, Architecture, Parallel and Distributed Computation, Theory, Artificial Intelligence, Data Bases and Software Engineering. The Department serves about 200 undergraduate and over 100 graduate students at the M.S. and Ph.D. level in Computer Science. The University is located near the shores of Lake Michigan, and is close to beautiful parks and pleasant residential areas.

Nominations/applications should be sent with the names of at least five references to:

Professor K. Vairavan
Chairperson-Computer Science
Department of Electrical Engineering
and Computer Science

University of Wisconsin-Milwaukee
The University of Wisconsin-Milwaukee is an equal opportunity/affirmative action employer.

UNIVERSITY OF MARYLAND

The Electrical Engineering Department of the University of Maryland College Park has openings for regular faculty positions in computer engineering. Candidates for the rank of Assistant Professor should have a high potential for both teaching and research. Candidates for the rank of Associate Professor and Full Professor should have distinguished records in research and a strong interest in educational programs. The department conducts research and educational programs in several areas of computer engineering. Although applications in all areas of computer engineering will be considered, operating systems, computer security, architectures for parallel computing, and VLSI are currently the highest priority areas for adding new faculty. Application, including resume, list of publications, list of references, identification of technical area for which the candidate wishes to be considered, should be sent to Dr. William W. Destler, Chairman, Electrical Engineering Department, University of Maryland, College Park, Maryland 20742. The University of Maryland is an equal opportunity, affirmative action employer.

PURDUE UNIVERSITY Computer Engineering Faculty Positions

The School of Electrical Engineering at Purdue University seeks outstanding candidates in all areas of Computer Engineering for research and teaching. Openings are for tenure-track faculty at all levels. Active research areas include artificial intelligence and expert systems; computer communication networks; computer vision; design automation tools; digital signal processor system design; distributed algorithms and databases; fault-tolerant and testable computing; microprocessor design; neural networks; parallel processing (architecture, algorithms, operating systems, compiling, languages, interconnection networks, and performance); robot vision, sensors and control; software engineering; speech processing; and VLSI architecture.

The School has 65 full-time faculty (26 in Computer Engineering), and over \$8.5M in funded research projects. In addition to the PhD, MSEE, and BSEE degrees, the School offers a BSCEE (Bachelor of Science in Computer and Electrical Engineering) which is accredited in both Computer Engineering and Electrical Engineering. Computing facilities available to the faculty include the Engineering Computer Network (including 17 VAX 11/780's running UNIX BSD 4.3, 1 Gould PN 9080, 4 Gould NP 1's, a CCI PN 6/32, and 225 Sun workstations), the Computing Center's Cyber-205 and ETA-10 supercomputers, a Symbolics LISP machine, and IBM 3090/180E, extensive graphics facilities, and numerous PC's. Parallel computing facilities include a 128-node Ncube hypercube, a 48 x 48 processor NCR-GAPP processor array, a 10 processor Transputer array, the 30-processor PASM Parallel Processor prototype that was developed and built at Purdue, and the

Computing Center's Sequent Balance 21000. Custom VLSI chip facilities include an IBM Master Image System. Equipment in the Robot Vision Lab and Computer Vision and Image Processing Lab includes a Puma 762 robot, a Cincinnati Milacron T3-726 robot, a K2A Cybermation mobile robot, and Gould/DeAnza, Comtal, Grinnell, and Imaging Technologies image processing systems. Support facilities provided by the School include a technical typing pool and a graphics illustrator.

Applicants must possess a doctorate degree. Send a resume, including a statement of teaching and research interests and a list of three (3) references, to: Head, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907. Purdue University is an Equal Opportunity/Affirmative Action employer.

COLORADO STATE UNIVERSITY

The Computer Science Department solicits applications for tenure-track and visiting faculty positions at all levels (subject to funding). Candidates for assistant professor need a PhD in computer science or computer engineering (at time of appointment) with promise for excellence in research and teaching; applicants for senior ranks must possess distinguished research records. We seek mainstream computer scientists in areas including architecture, artificial intelligence, graphics, languages and compilers, networks, software systems and engineering, and theory. In all areas we encourage applicants with interests in parallel computing, broadly defined: concurrent, distributed, vector, or systolic computing software, architectures, or applications. Salary is commensurate with rank and experience. New and visiting faculty will enjoy duties especially conducive to productive research.

The Department offers BS, MS, and PhD degrees. We have excellent cooperative research relations with industrial and government laboratories, and their people form a significant portion of our graduate student population. We operate numerous multi-user systems (HP, DEC, Sequent, ATT) and many workstations (Sun, Apple, ATT), all networked. University operations include a CYBER 205 vector supercomputer. Department personnel work in a pleasant, smoke-free environment.

Fort Collins is a growing community of 89,000 located along the foothills of the Rocky Mountains, 60 miles north of Denver. The climate is moderate—about 15 inches of precipitation and 290 days of sunshine per year. There are many cultural opportunities and year-round outdoor activities.

Send your curriculum vitae and names of at least three professional references to: Dr. R. R. Oldehoeft, Search Committee, Computer Science Department, Colorado State University, Fort Collins, CO 80523. Applications for August, 1989 will be considered April 1, 1989. The search will be extended if suitable candidates are not found.

Colorado State University is an EEO/AA employer. EO Office: 314 Student Services Building.

THE UNIVERSITY OF TENNESSEE SPACE INSTITUTE Faculty Positions in Computer Science

Applications are invited for tenure track senior level positions in all areas of computer science. A Ph.D. in computer science or a closely related area and a commitment to research and teaching are required. We are especially interested in faculty with established research and teaching records in computer science, refereed publications, experience in guiding Ph.D. students, and a record for obtaining grants and contracts. UTSI offers M.S. and Ph.D degrees in computer science with emphasis on applied artificial intelligence, expert systems and intelligent robotics.

UTSI is a multidisciplinary graduate and research institute offering degree programs and research in engineering, physics, applied mathematics and computer science. Emphasis is placed on research and graduate-level instruction. In addition to a VAX/780 and VAX/785, two Symbolics 3670 Lisp machines are available for research as well as linkage to computers at The University of Tennessee, Knoxville and supercomputers at various locations.

Rank and salary are open and commensurate with qualifications. Fringe benefits include group life and medical insurance, TIAA/CREF, and reduced tuition for dependents. UTSI occupies a scenic 365 acre lakeshore campus.

Send a detailed resume and names of three references to: Professor Moonis Ali, Chairman, Computer Science Program, The University of Tennessee Space Institute, Tullahoma, TN 37388. (Phone: (615) 455-0631, ext. 283).

UTSI is an AA/EEO employer.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY Faculty Position

Applications are invited for a tenure track position in the Department of Computer Science beginning Fall 1989. Applicants should normally have a doctorate in Computer Science or a closely related discipline. Rank and salary are competitive.

Rose-Hulman is a highly selective, primarily undergraduate, college of science and engineering. The 1300 students have an average SAT total score greater than 1200. The faculty are dedicated to quality teaching and continuing professional development. Additional information is available by e-mail from YoungF@VMS.Rose-Hulman.edu or by phone at (800) 248-7448 (Indiana (800) 552-0725).

Screening of candidates will begin Feb. 15, 1989 and continue until the position is filled. Send application and supporting documents to Prof. Frank H. Young, Rose-Hulman Institute of Technology, 5500 Wabash Avenue, Terre Haute, Indiana 47803. Rose-Hulman is an equal opportunity employer.

**UNIVERSITY OF NORTH TEXAS
COLLEGE OF ARTS AND SCIENCES
Department of Computer Sciences**

The Department invites applications for tenure-track faculty positions, pending budgetary approval. A Ph.D. in computer science or a related field and evidence of a strong commitment to research and teaching are required. Salary is competitive. Strong applicants in all areas of computer science are encouraged, but the Department has a special need for faculty in graphics, networks, and software development.

This growing department offers BS/MS/Ph.D. programs to approximately 450 undergraduate and 120 graduate majors. The University of North Texas is an emerging national research institution with excellent facilities to support both research and instruction in the vibrant and rapidly expanding Dallas-Fort Worth metropolitan area with over 700 regular faculty and over 24,000 students (one third graduate students).

Applicants should submit their resumes, including the names of three references, to Search Committee, Department of Computer Sciences, P.O. Box 13886, University of North Texas, Denton, Texas 76203.

UNT is an Affirmative Action/Equal Opportunity Employer.

TULANE

Tulane is a major private university, selective in enrollment and comprehensive in scope, with about 10,000 undergraduate and graduate students. The campus is located in a residential area of uptown New Orleans, one of America's most distinctive cities. Faculty benefits include dependent tuition, mortgage assistance, relocation expenses, 12% TIAA and typical insurance benefits.

The department computer network includes a Pyramid 9810 running UNIX, a VAX 11/780 running VMS, and several Sun workstations. This local area network is a subnet of the Tulane University network which is connected to SURANET, a regional subnet of NSFNET. Other major computing resources on the Tulane University network and accessible from Computer Science Offices include an IBM 3081KX mainframe. In addition, an IBM 4341 supports expert system research and instruction.

The department invites applications for faculty positions at all levels for the spring, 1989. Candidates should have a Ph.D. in Computer Science or a related area. Research excellence or demonstrated research potential and a commitment to quality teaching are required. Applicants for instructorships must possess a masters in Computer Science or ABD.

Applications should be directed to Dr. Johnette Hassell, Head, Department of Computer Science, School of Engineering, Tulane University, New Orleans, LA 70118. Applications must be received by March 16, 1989 for fall appointment. Tulane University is an Affirmative Action Equal Opportunity Employer.

**UNIVERSITY OF DELAWARE
Department of Computer and
Information Sciences**

Are you interested in joining the computer science faculty of a growing, dynamic department in an attractive university town within easy traveling distance to New York, Philadelphia, Baltimore, and Washington? The University of Delaware, centrally located on the East Coast, is recruiting for possible openings for tenure-track and visiting faculty positions in the Department of Computer and Information Sciences beginning September 1, 1989. The department has active groups working in artificial intelligence, networks, and symbolic mathematical computation. Special interest exists for candidates with research expertise in architecture, operating systems, parallel processing, databases, theory of computation, programming languages, or software engineering. Strong applicants in all areas of computer science are encouraged to apply.

A Ph.D. degree or its equivalent, and excellence in research and teaching are required. Salary and rank will be commensurate with the candidate's qualifications and experience.

The Department research facilities include various workstations (primarily Symbolics Lisp machines and SUN's) and facilities in a joint research lab shared with the Department of Electrical Engineering. The latter includes a Sequent Symmetry, VAX-8500, three VAX 780's and various other smaller machines. We are well connected to the major networks.

The Computer and Information Sciences Department offers bachelor, master, and doctoral degrees. Resources devoted to academic use in the University Computing Center include an IBM 3090-300 with a vector facility, a CDC Cyber 180, Unix machines (Vax 8650, Pyramid 98xe, Sun 4 with 128 MB memory, 12 Sun 3/60's), and more than 75 microcomputers (IBM PC-XT's, AT's and Macintosh's).

Candidates should send a curriculum vitae and the names of three references to Dr. David Saunders, Acting Chair, Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716. Positions are open until filled.

The University of Delaware is an equal opportunity, affirmative action employer. Applications from members of minority groups and woman are encouraged.

**TEXAS A&M UNIVERSITY
Endowed Chair in Microelectronics/
Computer Architecture**

Texas A&M University invites nomination of qualified individuals for an endowed chair in microelectronics, with specialization in the area of Computer Architecture. Candidates should have outstanding professional and personal qualifications, including national and international recognition for contributions in computer architecture. Nominations of qualified individuals from both industrial and academic backgrounds are solicited. The successful candidate will be expected to provide leadership in both research and

teaching in the computer architecture area. The academic rank will, of course, be Professor with tenure.

Texas A&M is making a major commitment to establishing and maintaining an outstanding program in Computer Science and Engineering. A substantial external contribution has enabled the creation of a heavily endowed chair in the microelectronics and computer architecture area as part of this effort. A new building is under construction for the Department. With the building the Department will receive a major allocation of funds for new equipment. In addition, the College is preparing to make substantial investments in facilities to support leading edge research in areas involving advanced computer architectures. There is also strong industry support for Department advances and ample opportunity for joint industry/university endeavors. The successful candidate can play a major role in the decisions to be made in these areas, and will have the opportunity to shape the direction of computer architecture research in the Department. Substantial resources will be made available for carrying out this work.

Related microelectronics research activities in the area of integrated circuits and systems include analog and digital integrated circuit design, VLSI system design, design automation, microprocessor architectures, and design for yield optimization. CAD facilities and software for IC design, layout and verification are available. A test and evaluation laboratory includes equipment for wafer handling, IC packaging, process characterization, and electronic test and evaluation.

Please send nominations to Prof. Richard A. Volz, Head, Department of Computer Science, Texas A&M University, 241 Zachry Engineering Center, College Station, Texas 77843. Initial screening will include applications received prior to April 1, 1989. Minorities and woman are particularly encouraged to apply.

**THE UNIVERSITY OF TEXAS
AT TYLER
Faculty Positions
Computer Science**

The Department of Mathematics and Computer Science at The University of Texas at Tyler anticipates faculty openings at all levels for the 1989-90 academic year. UTT is an upper-division, graduate level institution in eastern Texas with an approximate enrollment of 4000 students. At present the Department offers BS and MS degrees. Research computing equipment includes direct access to the Cray X-MP24 at UT Austin and an eight node Intel hypercube. Applicants with interests in neural nets, image processing, expert systems, data communications, architecture and distributed software systems will find interesting colleagues at UTT. Send resumes and a list of references to: Chair, Faculty Search Committee, Computer Science, The University of Texas at Tyler, 3900 University Blvd., Tyler, TX 75701 (214) 566-7402. UTT is an EO/AA employer—Male/Female.

SYRACUSE UNIVERSITY
The School of Computer and
Information Science

The School of Computer and Information Science at Syracuse University expects multiple openings in all areas of computer science at all professional levels. As part of our multi-year recruiting plan, we shall emphasize this year the core areas of Computer Science (Languages, Operating Systems, Software Design) and in particular, the sub-areas that relate to massively parallel systems. We shall, however, consider outstanding candidates in any area.

The School of Computer and Information Science operates as part of Syracuse University, a moderate-size private university committed to excellence in both teaching and research. We offer modern facilities, competitive compensation, and a congenial atmosphere in a scenic, central, and highly affordable part of the country.

Candidates should submit a statement of interest along with three to six names of reference and a resume to:

Faculty Search Committee
School of Computer and Information
Science

Syracuse University
313 Link Hall

Syracuse, New York 13244-1240

Application deadline is April 1, 1989.

Syracuse University is an Affirmative Action/Equal Opportunity Employer.

EMBRY-RIDDLE AERONAUTICAL
UNIVERSITY

Chairman

Department of Computer Science

The Department of Computer Science at Embry-Riddle Aeronautical University seeks applications for the position of chairman beginning in August 1989. We offer a bachelor's in CS and are developing plans to offer an EE degree. 7 full time faculty serve a growing undergraduate population of 175 majors.

The chair is responsible for maintaining academic excellence: recruiting and retaining faculty, and keeping the ACM78 based curriculum updated. A doctoral degree in CS or a related field is necessary; in the latter case, an MS in CS is highly desired. Candidates should have a teaching and scholarship profile commensurate with appointment to senior academic rank. An opportunity exists to increase the department's current involvement in over \$1 million of funded research activities at ERAU's Airway Science Simulation Laboratory.

Send a resume, and a letter detailing your qualifications and explaining why the job interests you, to:

Dr. R.O. Rogers, Chairman
Computer Science Chair Search
Committee

c/o Human Resources Department
Embry-Riddle Aeronautical University
Daytona Beach, FL 32014
Equal Opportunity Employer.

UNIVERSITY OF CALIFORNIA
AT RIVERSIDE
Positions in Computer Science

Applications and nominations are invited for tenured or tenure track positions in Computer Science beginning July 1, 1989 or later. The positions are open as to rank; candidates at all levels, including Full Professor, will be considered. Ph.D. and demonstrated excellence in research and teaching are required. All areas of Computer Science will be considered, including but not restricted to Programming Environment, Computer Systems, Theory of Computation, and Design and Development Automation. Salary and rank are determined by established criteria of the University of California. In addition to supplying a curriculum vita and a list of publications, candidates should either arrange for at least three letters of reference to be sent or submit the names of at least three references. The above materials should be sent to:

Professor Gerhard Gierz
Chair, Computer Science Recruiting
Committee

Department of Mathematics and
Computer Science

University of California
Riverside, CA 92521

The pool of candidates will consist of all those whose completed applications (consisting of a vita, a list of publications, and at least three letters of reference) are received by February 28, 1989. If the search has not been successfully completed by April 5, 1989, the pool will be enlarged to include all those whose completed applications are received between March 1, 1989 and April 5, 1989.

University of California, Riverside, is an Affirmative Action/Equal Opportunity Employer.

OREGON GRADUATE CENTER

How would you like to work in an academic environment with an active graduate program but no undergraduate teaching responsibilities? A place where research is a serious concern of everyone, with excellent facilities and a minimum of administrative red tape? The Oregon Graduate Center is such a place and we are expanding our program in Computer Science and Engineering. We seek faculty colleagues with experience in graduate education and with strong research interests. Some technical areas of particular importance to us are parallel computation, connectionist architectures, VLSI design and distributed database systems.

OGC is located in Portland, which remains the most affordable of the Pacific Coast's beautiful cities. It offers a place to combine a relaxed life style with an intense work style focused on the things you most want to do.

For more information about us, please send inquiries to: Prof. Richard Kiebertz, Chairman, Dept. of Computer Science and Engineering, Oregon Graduate Center, 19600 NW von Neumann Drive, Beaverton, OR 97006.

OGC is an equal opportunity employer.

THE UNIVERSITY OF TENNESSEE
SPACE INSTITUTE
Graduate Research Assistantships
and Post-Doctoral
Research Appointments

Applications are invited for graduate research assistants and post-doc research appointments in computer science, particularly in artificial intelligence, expert systems and neural networks. UTSI offers M.S. and Ph.D. degrees in computer science with emphasis on artificial intelligence.

The half-time research assistantships are granted on a nine-month basis with a stipend ranging from \$9,000 to \$11,500 plus waiver of a \$1,524 maintenance fee and a \$2,734 out-of-state tuition fee. Summer appointments may also be available. Post-doc salary is open. To receive an application or for further information please contact: Professor Moonis Ali, Chairman, Computer Science Program, The University of Tennessee Space Institute, Tullahoma, TN 37388 (Phone: (615) 455-0631, ext. 236).

UTSI is an AA/EEO employer.

THE UNIVERSITY OF TEXAS
AT ARLINGTON

The Department of Computer Science Engineering at The University of Texas at Arlington invites your application for tenure-track or visiting faculty positions in all areas of computer science and computer engineering. Rank is open. An earned doctorate or equivalent and commitment to teaching and scholarly research is required. Openings are expected for September 1989. Applications received prior to March 1, 1989 will receive full consideration. Interested persons should send a resume to Bill D. Carroll, Professor and Chairperson, Computer Science Engineering Department, P.O. Box 19015, The University of Texas at Arlington, Arlington, TX 76019. Phone 817-273-3785.

The University of Texas at Arlington is an Equal Opportunity Affirmative Action Employer.

RENSSELAER POLYTECHNIC
INSTITUTE

The Computer Science Department invites applications for tenure-track faculty positions at all academic ranks. Research, visiting and postdoctoral appointments may also be available. Applicants should have a Ph.D. in Computer Science (or a related area) and a commitment to excellence in teaching and research. Preferred research interests are parallel computation, database systems, computer graphics, computer vision, image processing, VLSI systems and symbolic computation; however, all areas will be considered. The department offers B.S., M.S., and Ph.D. degrees in Computer Science and has excellent computing facilities. Send resumes and at least three references to Joseph E. Flaherty, Chairman, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York 12180-3590. Rensselaer is an Equal Opportunity/Affirmative Action Employer.

NAVAL POSTGRADUATE SCHOOL Computer Science

The Computer Science Department invites applications for faculty positions at all levels. Our primary interests are in the areas of operating systems and programming languages. Our secondary interests are in the areas of visual data processing, graphics, and computer architecture (especially real-time and parallel-processing aspects of the three). Applicants should have a Ph.D. in Computer Science or a closely related field and be committed to high-quality teaching and research. Senior applicants must have distinguished research records. Appointments can begin at any time.

The Department offers M.S. and Ph.D. degrees in computer science, but no undergraduate degrees. Currently, 110 students are enrolled in the M.S. program and five in the Ph.D. program. Students are military officers or civilian employees of the Department of Defense and are fully supported by their sponsoring organization during their studies. Departmental facilities (supported by eight full-time computer professionals) include six instructional and research laboratories with extensive state-of-the-art equipment. Faculty normally teach for two quarters and perform research for two quarters per year. The Monterey-Carmel area provides a pleasant coastal climate and easy access to Silicon Valley companies.

Send a detailed resume, an abstract of your best recent research, and letters of reference to Faculty Search Committee, Computer Science, Code 52, Naval Postgraduate School, Monterey, CA 93943 (phone (408) 646-2449). An Equal Opportunity/Affirmative Action Employer.

PORTLAND STATE UNIVERSITY

The Department of Electrical Engineering currently has a tenure-track faculty position at the Assistant or Associate Professor level. Applicants must have an earned doctorate and demonstrated research expertise in an electrical engineering field. The areas of particular interest involve all aspects of Computer Engineering, but concomitant teaching and research activities would be desirable in other electrical engineering disciplines such as Controls and Robotics, Electromagnetics, Lasers, Communications, Integrated Circuit Design, Solid State Electronics, etc. Advanced Electrical Engineering research laboratories in the University's Portland Center for Advanced Technology include a VLSI design center, integrated circuit laboratory, computer vision and graphics laboratories, and laser laboratories. Faculty members have access to a departmental network of VAXes and Sun workstations as well as university IBM and Gould computers. Responsibilities include undergraduate and graduate teaching, development of sponsored research, and interaction with local industry.

Portland State University is one of the three major universities in the Oregon State System of Higher Education. It is located in Portland, Oregon, with easy access to the

well known recreational opportunities of Oregon's mountains, rivers, and coast. The Department of Electrical Engineering offers B.S., M.S., and Ph.D. degrees in electrical and computer engineering. The Department has 10 full-time faculty and over 300 students admitted to its upper division and graduate programs.

Portland has a rapidly-growing computer and electronics industry including Tektronix, Intel, Hewlett-Packard, FPS Computing, NCUBE, Electro-Scientific Industries, Sequent Computer Systems, Metheus, Mentor Graphics, Lattice, and many others, which permits close industry-university interaction.

Rank and salary are commensurate with qualifications and experience. Send an application, including a resume listing the names of three references to: Dr. Rolf Schaumann, Chair, Department of Electrical Engineering, Portland State University, Portland, Oregon 97207-0751. Phone: (503) 464-3806. Applications will be accepted until March 15, 1989, or until suitable candidates are identified.

Non-U.S. residents must state their visa status. Portland State University is an equal opportunity/affirmative action employer.

COLGATE UNIVERSITY Hamilton, New York 13346 Department of Computer Science

We invite applications for a half-time leave replacement position for 1989-1990. This would be an ideal position for someone who wants to combine some teaching with research during a sabbatical. The position may be filled by someone teaching half-time for the year or full-time for one semester (two courses with labs or three without).

Colgate is a quality liberal arts college with a first-rate computer science program. The department has five faculty with the following research interests: theory of computation and programming language semantics, parallel implementations of functional programming languages, temporal reasoning in natural language processing, graphics and chaos, and discrete event simulation on parallel computers. The Computer Science Department has an introductory lab equipped with sixteen PC and PC/AT clones, and an upper-level/research lab with the following equipment: a VAX 750 running BSD 4.3 Unix, a TI Explorer, four 17-node transputer-based parallel computers (which can be configured as a single 68-node network), five Amiga PC's, and PC/AT clones in every faculty office. Over the summer the department will install a network of either Sun or NeXT workstations. The faculty offices and laboratory machines are connected on an ethernet. We are members of both CSNet and BitNet.

Applicants should send a resume and the names of three references to: Chris Nevison, Chairman, Department of Computer Science, Colgate University, Hamilton, NY 13346. All applications received before April 1, 1989 and after that until the position is filled, will be considered. Colgate University is an affirmative action/equal opportunity employer.

REASONING WITH KNOWLEDGE

The Reasoning Architectures Group at MCC is exploring techniques for reasoning with knowledge to solve a variety of problems, particularly in the areas of planning, design, and natural language comprehension. Recent work by the group has produced results in hybrid expert system architectures, high-performance inferencing, metareasoning, truth maintenance, machine learning, and lexical semantics.

Some of our newer efforts are in defeasible reasoning, dynamic planning, qualitative reasoning, cooperative problem solving, reflective reasoning, and knowledge-directed semantic interpretation.

We are now hiring bright people with new ideas who work well with other researchers. We want Ph.D. level researchers who can make strong contributions, help direct the research, and be part of a growing research community. We offer a highly collaborative research environment that includes other research groups at MCC (who are working in the related areas of knowledge representation, intelligent interfaces, logic programming languages, and object-oriented databases) as well as research and application groups within our shareholder companies.

MCC employees enjoy very competitive salaries and a top rate computational environment in a beautiful setting in the sunny Southwest. Austin is home to the University of Texas and to other research centers, including Schlumberger, Lockheed, Sematech, and 3M.

For more information contact: Elaine Rich, Director, Artificial Intelligence Laboratory, MCC, 3500 West Balcones Center Drive, Austin, Texas 78759, or, rich@mcc.com.

FAIRLEIGH DICKINSON UNIVERSITY Assistant Professor of Computer Science

Florham-Madison Campus: A full-time, tenure-track position starting September 1, 1989. The Florham-Madison Campus of FDU is located in the Northern New Jersey center of corporate research and development activity and within 35 miles of New York City. Responsibilities include teaching of undergraduate and graduate computer science courses, student advising, scholarly research, and participation in the development of growing computer science program.

Applicants should have a Ph.D. in computer science or be within one year of completion of Ph.D. requirements. Teaching experience preferred. Salary competitive and commensurate with qualifications and experience. Letter or application and resume should be sent to: Professor Peter Falley, Chair, Department of Math/CS/Physics, Fairleigh Dickinson University, Madison, N.J., 07940. Applications will be received until March 1989, or until the position is filled. Women and minorities are encouraged to apply. FDU is an equal opportunity/affirmative action employer. M/F

UNIVERSITY OF KENTUCKY

The Center for Robotics and Manufacturing Systems, a unit of the College of Engineering at the University of Kentucky invites applicants for positions at the associate and full professor levels. A doctorate in engineering or computer science is required. Applicants from academia or industry must have experience in manufacturing processes, factory automation, and demonstrated capability to achieve research sponsorship with federal agencies or industry. Major responsibilities are to conduct and supervise research, work with industrial partners, and teach courses including those within a graduate-level Manufacturing Systems Engineering curriculum. Desired areas of research include sensor-based mechanical assembly, rapid prototyping of molded plastic parts, surface mount assembly and inspection, and computer integrated manufacturing.

These positions are a result of major support by the Commonwealth of Kentucky and local industry. A new 68,000 sq. ft. building will provide more than 60 offices and laboratories for faculty and staff. The building will be extensively equipped for CAD/CAM/CAE, flexible machining, assembly, material handling, injection molding, and electronics manufacturing. Senior manufacturing engineers on the CRMS staff assist faculty to market and manage grants and industrial projects. Local industries include Clark Equipment, GE, IBM, Litton and Whirlpool. Located in the heart of the Bluegrass, the Lexington area offers a mild climate, scenic beauty, and quality living.

The appointment for these positions, available July 1, 1989, will be with an academic department of the University of Kentucky and is subject to university criteria for promotion and tenure. A limited number of tenure-track assistant professor appointments and full-time research staff positions also may be available. Nominations or applications, with full resume and the names of four references, should be sent to Chairman, CRMS Search Committee, College of Engineering, University of Kentucky, Lexington, KY 40506-0046.

THE WILLIAM CRAIG NYSTUL CHAIR IN COMPUTER SCIENCE AT THE UNIVERSITY OF WEST FLORIDA

Nominations and applications are invited for the William Craig Nystul Chair of Computer Science at The University of West Florida. This is an endowed position funded by private contributions and by a matching grant from the State of Florida Eminent Scholars Trust Fund. The Chair resides in the Division of Computer Science, an academic unit reporting directly to the Vice President for Academic Affairs.

The Division of Computer Science offers the bachelors degree in three options—Computer Information Systems, Systems and Control Engineering, and traditional Computer Science. At the graduate level, the masters degree is offered with options in Systems and Control Engineering and Soft-

ware Engineering. The Division of Computer Science is housed in a new computing center containing an IBM 4381 mainframe, a Sequent Symmetry parallel processing minicomputer, several Motorola development systems using both UNIX and VER-5Ados development software, and a host of Macintosh, IBM, and Zenith microcomputers. The Computer Science faculty totals seventeen, with some growth expected over the next several years.

Applicants and nominees should have national stature in the academic and/or practicing computer science profession and an earned doctorate in computer science or other appropriate field. Candidates should have substantial tangible evidence of research and scholarly activity in the field of computer science, preferably in the areas of artificial intelligence/expert systems or software engineering.

The successful candidate will be expected to provide leadership and direction in teaching, research, and professional service. However, he or she will have wide discretion with regard to how this leadership is provided. Salary is negotiable based upon qualifications.

Nominations and applications will be accepted until the position is filled and should be sent to: Dr. Theodore F. Elbert, Chairman, Eminent Scholar Selection Committee, Division of Computer Science, The University of West Florida, Pensacola, FL 32514-5750.

The University of West Florida is an equal employment opportunity/affirmative action employer.

SYSTEMS ENGINEER

Systems Engineer for Cleveland, Ohio area firm to design artificial intelligence software systems for business applications using general systems approach to accommodate spectrum of alternative theories and assumptions; design for judgmental and subjective aspects; provide systems approach for assimilating change to parameters and structure during analysis or decision-making process (not only before or after); assist in preparation of models of large scale economic, demographic, resource and technological systems for different regions, areas and governments; employ mathematical control theory and systems analysis in preparing Large Scale Systems Analysis; use large scale systems analysis to create World Systems Models. No experience required. Must have M.S. in Systems Engineering with research work or thesis in field of Large Scale Systems Analysis, specifically in World Systems Models; and undergraduate and graduate coursework in the following: Systems Engineering (9 credit hrs. including 6 in study of topic signals and systems), System Simulations (3 credit hrs.), Decision Theory (6 credit hrs.), Optimization of Systems (6 credit hrs.), and Control Engineering (4 credit hrs.). 40 hrs./wk, 8am-5pm. \$29,700/yr. Qualified applicants reply immediately with resume to R. Lechler, JO# 1084405, Ohio Bureau of Employment Services, P.O. Box 1618, Columbus, OH 43216.

WRIGHT STATE UNIVERSITY Department of Computer Science and Engineering Dayton, OH 45435

Applicants are invited for tenure-track and visiting faculty positions at all levels. Successful candidates for tenure-track positions should have a Ph.D. in computer science, computer engineering, or equivalent background and have demonstrated forward looking and creative research. Further desired attributes include: capability to direct Ph.D. candidates in computer science or computer engineering and the ability to acquire funds and/or direct research projects. All technical areas will be considered but AI related areas including machine learning and neural networks are of particular interest. Rank and competitive salaries are determined by qualifications and experience.

The university is located in a high technology environment among industrial/military research and development facilities including Wright-Patterson Air Force Base and NCR. The graduate program has excellent offices and computing facilities in the WSU Research Center located in a fast growing associated state-assisted research park fostering basic and applied industrial/military/university research. The Center for AI Applications and the Edison Materials Technology Center (EMTEC) funding organizations are located in the same building.

Department strengths include a large faculty, extensive laboratory facilities including graduate laboratories in AI, optical computing, neural networks, and robotics, established research programs, industrial/military support, degree programs in both computer science and computer engineering, and large student populations at graduate as well as undergraduate levels.

Please submit a detailed resume including names of three references to: CSNET address: amcaulay@wright.edu or Alastair D. McAulay, NCR Distinguished Professor and Chair, Department of Computer Science and Engineering, Wright State University, Dayton, OH 45435. Reviewing for positions will begin immediately and continue monthly until positions are filled or until April 1, 1989.

An equal opportunity/affirmative action employer.

CLARKSON UNIVERSITY

The Clarkson University Mathematics and Computer Science Department is interested in hiring a faculty member in the areas: (a) Computer science, in particular theoretical computer science; (b) computational mathematics; (c) pure mathematics, in particular differential geometry and algebraic topology. Teaching load is two courses per semester. Salary and benefits are outstanding.

Interested applicants should send resume and three letters of recommendation to Professor A.S. Fokas, Chairman, Department of Mathematics and Computer Science, Clarkson University, Potsdam, NY 13676.

Clarkson University is an Affirmative Action/Equal Employment Opportunity Employer MFVH (Minority, Female, Veteran, Handicap).

**RENSSELAER POLYTECHNIC
INSTITUTE**
Faculty Positions
**Department of Electrical, Computer,
and Systems Engineering**

Applications are invited for tenure-track faculty positions at all levels. Specific areas of interest include: (1) computer engineering, architecture, and performance evaluation, particularly related to parallel image processing algorithms and architectures; (2) computer and communications networks, architectures and protocols; (3) solid-state and optoelectronic devices, interconnects, and systems. The ECSE Department is the largest academic unit at Rensselaer and has a rich tradition of research and education. The department is seeking to add faculty who bring an innovative approach to research and teaching. Active programs in computer engineering, solid-state electronics and integrated circuit design, control systems, robotics and automation, information and decision systems, communications and signal processing, electronics and circuits, and fusion plasma systems contribute to a dynamic research environment. In addition to the extensive research facilities of the department, there are opportunities to initiate or participate in interdisciplinary research programs in one of the major research centers of the School of Engineering, including the Center for Integrated Electronics, the Center for Rensselaer Design Research, the Center for Manufacturing Productivity and Technology Transfer, and the NASA Center for Intelligent Robotic Systems in Space Exploration. New faculty are eligible for special arrangements including summer support, equipment, graduate student support, and reduced teaching load in order to encourage growth of their research programs. Applications or requests for more information should be directed to:

Dr. Arthur C. Sanderson, Chairman
Electrical, Computer and Systems
Engineering Department
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

RPI is an affirmative action/equal opportunity employer.

PRINCETON UNIVERSITY
**Department of Mechanical &
Aerospace Engineering**
Professional Technical Staff Member
Electrical Engineer

The Navier-Stokes Supercomputer Laboratory is seeking an electrical engineer with experience in computer circuit design and layout. BSE in electrical engineering with CAD experience preferred. Knowledge of schematic capture systems and the CALAY Printed-Circuit CAD station helpful, but not required.

Department of Mechanical
and Aerospace Engineering
Code EAW/MEL
Room D212 Engineering Quadrangle
Princeton University
Princeton, NJ 08544

Princeton University is an Affirmative Action/Equal Opportunity Employer, M/F.

OREGON STATE UNIVERSITY
Department of Computer Science

The Department of Computer Science invites qualified applicants for tenure-track Assistant, Associate and Full Professorships. Specialization in programming languages and systems, information-based systems, software engineering, theoretical computer science, computer architecture, artificial intelligence or computer graphics is desirable but all qualified applicants will be considered. Applicants should have completed or expect to complete all requirements for a Ph.D. in computer science or a closely related field and should have demonstrated research and teaching potential. Candidates for senior positions should have an established research reputation. Review of applications will begin November 1, 1988, and will continue until the positions are filled. Please send resume, including the names of three references, to: Walter G. Rudd, Chairman, Department of Computer Science, Oregon State University, Corvallis, OR 97331.

Oregon State University is an equal opportunity affirmative action employer and complies with Section 504 of the Rehabilitation Act of 1973. OSU has a policy of being responsive to the needs of dual-career couples.

CASE INSTITUTE OF TECHNOLOGY
Nord Professorship in
Computer Engineering and Science

The Department of Computer Engineering and Science at Case Institute of Technology is seeking a nationally recognized scholar and researcher to fill the NORD Professorship. This position was recently established by the donation of one and a half million dollars, which will provide outstanding professional opportunities and a highly competitive salary, together with additional funds for travel, graduate student support and equipment. The qualifications include a Ph.D. in computer science, computer engineering or closely allied fields, and an ability to establish and develop external support for a nationally recognized research program in computer science/computer engineering. Another requirement of this position is excellent teaching at both the undergraduate and graduate levels. We invite applications from senior faculty at both the associate professor and full professor levels.

CWRU is a private university with a total enrollment of 8,400, of which 5,100 are graduate and professional students. The Engineering School of Case Institute of Technology is among the top ten engineering schools in terms of research funding per faculty member and undergraduate student quality. The University campus is the hub of the pleasant area known as University Circle, an incorporation with neighboring cultural centers and museums, about five miles from downtown Cleveland.

The Department of Computer Engineering and Science has 15 faculty positions, and a graduate student body of 140 students, 50 of which are in the Ph.D. program. Departmental facilities are based upon an ethernet local area network, connected to INTER-

NET, which supports a UNIX operating system and about 30 SUN workstations, color graphics display terminals, together with several printers. In addition, faculty and students participating in the Center for Automation and Intelligent Systems Research have access to the Center's computing facilities.

Applicants should submit their curriculum vitae and names of at least five references to: Lee J. White, Chairman, **Department of Computer Engineering and Science**, Case Western Reserve University, Cleveland, Ohio 44106; INTERNET: leew@alpha.ces.cwru.edu; applicants may wish to provide at most three reprints of their most important publications.

An equal employment and affirmative action employer.

SOFTWARE ENGINEER

Software Engineer for medical diagnostic image system manufacturer in NE Ohio. Conduct research and development of software for computerized medical image system. Responsible for: 1. Transferring reconstructed x-ray scan data of human body to display system designed for reviewing by physicians and radiologists for diagnostic purposes. 2. Developing computer graphics primitives to implement display function. 3. Assisting in developing Data Base for image file archiving and patient record management. No experience necessary. Must have MS degree in Computer Science. Applicant must have undergraduate degree in chemistry. Graduate coursework must have included "C" Language, File and Data Base, Information Retrieval, Computer Graphics, Digital Computing Method, Matrix Theory, Data Structure, Advanced Analytical Chemistry and Principles of Quantum Chemistry. 40hrs/wk, 8am-5pm, \$30,685/yr. Qualified applicants reply immediately with resume to A. MacLean JO# 1084414, Ohio Bureau of Employment Services, P.O. Box 1618, Columbus, Ohio, 43216.

ILLINOIS WESLEYAN UNIVERSITY
Computer Science

Illinois Wesleyan University seeks candidates for a tenure-track computer science position at the Assistant Professor level beginning August, 1989. Applicants should possess an M.S. in Computer Science and a strong interest in teaching at the undergraduate level. A Ph.D. in a related field is highly desirable.

Illinois Wesleyan University is a strong undergraduate institution that attracts highly qualified students. It is within easy driving distance from Chicago and St. Louis. The university is expanding its computer facilities and computer science curriculum.

Applicants should send a resume, and three letters of recommendation to:

Dr. Susan Anderson-Freed
Coordinator, Computer Science
Illinois Wesleyan University
Bloomington, IL 61761

Women and minorities are encouraged to apply. Equal Opportunity Employer.

UNIVERSITY OF KANSAS
Electrical and Computer Engineering

The Electrical and Computer Engineering Department of the University of Kansas seeks to fill several tenure-track and visiting faculty openings, available beginning August 1989, or as negotiated. Applicants should have teaching and research interests in Artificial Intelligence, Software Engineering, or Computer Architectures. Applicants with added experience and/or research record in VLSI will be given first preference. New faculty will be expected to perform teaching and curriculum development activities, and to participate in on-going research activities as well as to develop their own research. Rank and salary will be commensurate with qualifications and experience. A doctorate is required. Resumes should be sent to Dr. James R. Rowland, Chairman, Electrical and Computer Engineering Department, University of Kansas, Lawrence, Kansas 66045-2228. Phone: (913) 864-4620. E-mail: jrowland@volta.eca.ukans.edu. The University of Kansas is an equal opportunity, affirmative action employer.

McMASTER UNIVERSITY
Communications Research Laboratory
(Telecommunications Research
Institute of Ontario)
and Department of
Computer Science and Systems
Faculty Position

The Communications Research Laboratory (CRL), through its affiliation with the Telecommunications Research Institute of Ontario, and in partnership with the Department of Computer Science and Systems is well along in establishing a unique "computing environment for advanced research in signal processing."

In keeping with this objective the Department invites applications for a faculty position from candidates who have a Ph.D., demonstrated teaching ability, and a proven research record in one or more of the general areas of parallel system architecture, pattern recognition, and computer vision.

The successful candidate will also be a member of the CRL with the privilege of access to advanced research facilities and the opportunity to conduct world-class research under the umbrella of TRIO a "Centre of Excellence." Departmental responsibilities include teaching at the undergraduate and graduate levels and the supervision of graduate students. Salary is competitive for a tenure-track position and commensurate with qualifications and experience.

Please provide a curriculum vitae and arrange for three letters of reference, to be sent to (or request additional information from):

Dr. S. Haykin, Director
Communications Research Laboratory
OR

Dr. G.L. Keech, Chairman
Computer Science and Systems
McMaster University
Hamilton, Ontario
L8S 4K1

Deadline for Applications: March 31, 1989.

THE UNIVERSITY OF TRONDHEIM
THE NORWEGIAN INSTITUTE
OF TECHNOLOGY
Department of Electrical Engineering
and Computer Science

The Department of Electrical Engineering and Computer Science has four full-time faculty positions available at the assistant/associate professor level. The areas of specialization are switching systems, software engineering, basic program systems and computer architecture. For all positions a Ph.D. in the relevant area is required, with evidence of strong research accomplishment or potential. Interest and ability in teaching graduates and undergraduates is necessary. The appointed assistant/associate professors will be expected to participate in the externally funded research activities of the department. The computer equipment is mainly PCs and UNIX workstations (SUNS and VAXes).

The Norwegian Institute of Technology has 5600 students and is part of the University of Trondheim. With the associated SINTEF research organization, teaching and research staff in the field of information technology number approx. 300 scientists. The city of Trondheim has 135,000 inhabitants. There are good state schools in the area, also an English primary school. English is widely spoken, both at the Institute and in the community. There are excellent opportunities for outdoor activity in the vicinity including skiing, fishing, hiking and cycling.

Applications close on 1 April 1989.

Further information about the positions and the teaching programme may be obtained by contacting Professor Peder J. Emstad, telephone +47-7-594326, net address on BITNET: emstad%vax.elab.unit.uninett@norunix, at The Division of Computer Systems and Telematics.

Send your application to the University of Trondheim, Norwegian Institute of Technology, Personell Section, N-7034 Trondheim, Norway.

SAINT MARY'S UNIVERSITY
Computing Science

Saint Mary's University, Department of Mathematics and Computing Science invites applications for a tenure-track position at the rank of Assistant Professor effective September 1, 1989. Applicants should have a doctoral degree in Computing Science; however, candidates with doctoral degrees in other areas with a strong background in Computing Science may be considered. Duties will include teaching and research, and candidates with a demonstrated ability to teach undergraduate Computing Science will be given preference. In accordance with Canadian immigration requirements, this advertisement is directed in the first instance to Canadian citizens and permanent residents; however, all qualified candidates are strongly encouraged to apply. Applications, including the names of three references, should be sent to: Dr. M.T. Kiang, Chairperson, Department of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia, Canada B3H 3C3.

UNIVERSITY OF CALIFORNIA,
LOS ANGELES
COMPUTER SCIENCE DEPARTMENT

The Department of Computer Science at the University of California, Los Angeles, invites applications for tenure-track positions at the Assistant Professor level in Computer Science beginning in July 1989. Applicants should possess the Ph.D. in Computer Science by July 1989. Applications are also welcome from highly distinguished candidates at the senior level.

Quality is our key criterion for selecting applicants. We expect them to have a strong commitment to both research and teaching and an outstanding record of research for their level. It is important that they exhibit strong potential for continued excellence in university research.

We seek applicants in any mainstream area of Computer Science and we particularly welcome those with research strength in software related areas.

Interested applicants should send a letter of application, a resume, and the names of four references to:

Professor Wesley W. Chu, Chair
Computer Science Department
Attn: Ms. June Myers
Boelter Hall 3732

University of California, Los Angeles
Los Angeles, CA 90024-1596

The University of California is an Affirmative Action/Equal Opportunity Employer.

CARNEGIE MELLON UNIVERSITY
Faculty Position in Field Robotics/
Construction Automation/Sensing

Carnegie Mellon University, Department of Civil Engineering, invites applications for a tenure-track position as an Assistant Professor to begin in September 1989. Candidates for this position should have an earned doctorate and research experience in field robotics, remote sensing or construction automation, and must be able to interact with other faculty working in robotics, computer-aided engineering and construction project management. Candidates should have strong analytical capabilities (preferably in mechanics or control) and/or experience with microprocessor-based systems, sensors and instrumentation. While a background in engineering is necessary, training in civil engineering is not a requirement. Responsibilities include teaching undergraduate and graduate courses, supervising graduate students and developing a funded research program. Current field robotics/construction automation activities (including fundamental studies in mobility, vision, force sensing, and system integration) in the department are well established, and many opportunities exist for professional and intellectual growth. The department also has close ties with the university's Robotics Institute, and a joint appointment with the Robotics Institute is possible. Interested persons should contact Professor Chris Hendrickson, Faculty Search Committee, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA 15213-3890. Carnegie Mellon is an equal opportunity/affirmative action employer.

BOOK REVIEWS

Editor: Wiley McKinzie, School of Computer Science and Technology, Rochester Institute of Technology, Rochester, NY 14623; Compmail, w.mckinzie; CSnet, wrm@rit

Principles of Database and Knowledge-Base Systems: Volume I

Jeffrey D. Ullman (Computer Science Press, Rockville, Md., 1988, 631 pp., \$41.95)

Jeffrey Ullman is one of a handful of computer scientists who have consistently produced outstanding books. Among my favorites is *Principles of Compiler Design* (Addison-Wesley, 1977), which he coauthored with Alfred Aho. Long regarded as a classic on the subject of compiler construction, this book is more often referred to as the Dragon Book — because of its cover — than by its title.

Principles of Database and Knowledge-Base Systems will be known as the Baseball Book because it discusses databases, object bases, and knowledge bases with the same care and precision that the Dragon Book discussed compiler design. Like the Dragon Book, the Baseball Book's cover cleverly makes a pictorial play on words with a baseball diamond.

The Baseball Book is the first in a planned two-volume set designed to replace Ullman's earlier *Principles of Database Systems*. While much of the same content appears in this new volume, the material has been substantially rearranged to reflect current trends. Only a discussion of optimization and universal relations is missing from the new volume. These topics will form the focus of the planned second volume. New material introducing knowledge-base systems also appears in the first volume, with applications of these systems reserved for the second volume.

Ullman divides his discussion into 10 chapters that cover basic terminology, data models, database languages, physical data organization, relational database design, and protection in both distributed and nondistributed systems. As in the earlier volume, Ullman presents each topic from the perspective of a patient, caring teacher who enjoys his subject matter and believes that humor contributes to the learning process. Ullman always couples his detailed explanations with examples. For topics with mathematical underpinnings, he provides both theorems and proofs.

Frequently, explanations are translated into novel algorithms designed to show readers how to apply the concepts practically.

For example, in his discussion of the relational data model, Ullman couples the mathematical premises of the model with a superb synopsis of relational algebra. A small database, the Yuppie Valley Culinary Boutique, is used to translate the theoretical model into reality. This database appears in the discussion of the other models, providing readers with a sense of continuity.

Like its predecessor, the Baseball Book's coverage of database theory and applications is comprehensive. The earlier *Principles of Database Systems*, however, provided a very limited number of exercises to test the reader's understanding. The Baseball Book has corrected this flaw by including not only extensive chapter exercises but also an annotated bibliography of each major subject. Thus, the Baseball Book is an ideal text for advanced undergraduate or graduate students. It is also a superb resource for professional database designers and applied mathematicians. The book does presume a mathematically sophisticated reader, and since Ullman frequently draws upon relevant material from operating systems, algorithm analysis, and logic programming, a strong background in these areas is highly desirable.

For readers familiar with *Principles of Database Systems*, I will describe briefly the new features of the Baseball Book. Striking among the differences is the classification of the hierarchical and network models under the rubric of "object bases," also referred to as "object-oriented database systems," and the combined coverage of languages embodying these models in a single chapter. The treatment of hierarchical and network languages (for example, the database definition language proposed by the Database Task Group of the Conference on Data Systems and Languages) remains essentially unchanged from the earlier book, but Ullman does discuss Opal, a new object-oriented language that provides the user interface for the

Gemstone database system.

This reorganization reflects a change in orientation that is apparent in the introductory chapter. In addition to discussing general terminology, Ullman uses his preliminary remarks to present the reader with a new classification of databases. Rather than using the standard distinction of hierarchical, network, and relational databases, Ullman classifies databases as "object-oriented" or "value-oriented." This contrast focuses on a database's ability to distinguish between objects that appear to be the same.

For example, assume that we have a file with two fields: "name" and "occupation." Also assume that this file has two occurrences of "Clark Kent" and "Superman." Object-oriented databases (network and hierarchical models) possess the ability to distinguish between these two occurrences, value-oriented ones (relational and knowledge-base systems) do not. This taxonomy allows Ullman to incorporate material effectively on the new knowledge-base systems.

The jewel of the book is, however, Ullman's analysis of the theoretical foundation of knowledge-base systems. Beginning with a variant of Prolog suitable for database applications, referred to as Datalog, Ullman deftly presents the rudiments of logic programming. He meticulously discusses Horn clauses and develops algorithms to determine safe rules (that is, those that produce results in a finite time) for nonrecursive, recursive, and negated clauses. The chapter concludes with a formal proof demonstrating that Datalog programs have the same expressive power as relational algebra expressions and formulas in relational tuple calculus or relational domain calculus.

In conclusion, *Principles of Database and Knowledge-Base Systems: Volume I* embodies the forefront of research on database systems. The incorporation of knowledge-base systems into database analysis is rare, and Ullman's coverage is brilliant. I eagerly await the release of the second volume.

Susan Anderson-Freed
Illinois Wesleyan University

Atanasoff: Forgotten Father of the Computer

Clark R. Mollenhoff (Iowa State University Press, Ames, Iowa, 1988, 274 pp., \$24.95)

John Vincent Atanasoff possesses the qualifications necessary to obtain the love of his equals. He is open and punctual in his dealings, candid in his judgment, blameless in his manner, and always ready to communicate. Indeed, this paradoxical physicist designed and, together with his Iowa State graduate assistant, C.E. Berry, built a working version of an electronic digital computer before the US entered World War II.

The machine, called ABC (Atanasoff-Berry Computer), could invert 29-by-29 matrices. It did not have the important applications of the British Colossus machine, which was used by early 1944 to break the top-secret Nazi Enigma Code in near real time. Nor did it have important applications to ballistics and fusion-weapons feasibility studies like the Eniac had after 1946. But ABC worked and predated them.

Here is the sordid story of many people, all too human, caught up in the emotional fury of a great technological invention, the electronic digital computer. Similar stories, but with unique twists, exist for the telephone, the automobile, the airplane, the structure of DNA, and the laser, among other discoveries and inventions. In all these cases, the behavior of the principals is not as celestial as one might expect but, rather, very terrestrial.

Mollenhoff, a Pulitzer Prize-winning investigative journalist, develops the convoluted relation between Atanasoff and John Mauchly in great documented detail. Many founders of our computer world are mentioned, including Brainerd, Burks, Bush, Caldwell, Eckert, Goldstine, Metropolis, Turing, and von Neumann. Indeed, it has been 15 years since Judge Larson's formal decision in favor of Honeywell (Atanasoff) and against Sperry Rand (Mauchly).

Topics discussed and documented in the 22 chapters include the need to set the record straight, the war years, Eniac, and charges of patent fraud and dissembling. Interestingly, it turns out that Atanasoff's crucial ideas for the electronic digital computer came together in Illinois. (This shouldn't surprise Illiac users!)

Two other important and independent works that support the idea that Atanasoff invented the electronic digital computer are Allan R. Mackintosh's "The First Electronic Computer" (*Physics Today*, Mar. 1987, pp. 25-32) and Alice and Arthur Burks' *The First Electronic*

Computer: The Atanasoff Story (Univ. of Michigan Press, 1987).

Mollenhoff's book raises as many questions and puzzles as it answers. How can a network of fraud and deception infiltrate an intellectual environment?

This book leads inexorably to the conclusion that ethical behavior is not optional in engineering and science.

Why did Iowa State fail to patent the ABC? Was C.E. Berry murdered or did he commit suicide in 1963? Ironically, the ABC was dismantled by an Iowa

State graduate student — who went on to become head of the Computer Science Department — based on instructions from his senior professor.

This book leads inexorably to the conclusion that ethical behavior is not optional in engineering and science. Yet, today, there are documented cases of serious wrongdoing. I cannot believe that engineers are either unconcerned or unashamed about ethical problems in their profession. Engineers must be the first to confront misconduct, whether it is fabrication, falsification, plagiarism, or deception. For these reasons, I highly recommend this well-written book to students and teachers of computer engineering and science as well as to general readers.

Albert A. Mullin
US Strategic Defense Command

Software Communications Skills

Robert L. Glass (Prentice Hall, Englewood Cliffs, N.J., 1988, 513 pp., \$27)

Many scientific and technical people find it difficult to communicate the results of their work to peers, managers, and the user community in a form that can be easily understood. The problem is particularly pervasive in the software community, where communication skills are required in walkthroughs, project management, and documentation. Glass' book, *Software Communications Skills*, attempts to throw a communication lifeline to computer professionals.

Despite its 513-page length, this is actually quite a slight book. The first 175 pages are advice, while the remaining pages are reprints of useful but generally available software documentation standards. Since these standards only serve to fill out the book and raise its price, this review will center on the rest of the book.

Software Communications Skills is an odd mixture of generalizations about communicating and specializations to the software environment. Chapter 1 is an introduction, Chapter 2 consists of homilies on good communication, Chapter 4 comprises six pages on communicating with a nontechnical audience, and Chapter 5 is an aside on office automation that never mentions desktop publishing explicitly.

The three sections of Chapter 3 entitled

"Project Start-up," "Project Implementation," and "Project Wind-up" are the book's centerpiece. I wish I could recommend these, but I can't. The subsections provide sequential lists and advice on what documents to prepare at what stage of the software life cycle. Checklists abound. Sets of generic questions are sprinkled liberally throughout, as are outlines taken from the standards. For example, the seven questions in the section on creating the user manual start with "What capabilities does the product offer?" and "How and where can the product be accessed?" and end with "What error diagnostics are there and how should the user respond to them?" Although useful as checklists, the questions and outlines offer little in the way of substance.

There are no extensive examples of the content of any of the documents described. If you don't have the aesthetic judgment or experience to tell the difference between good and bad documentation, there is no way you can learn it here.

In short, this book is basically an expanded set of lecture notes for a one- or two-day seminar, such as those given at local hotels by roving lecturers on writing, supplemented by handouts of standards from the Government Printing Office.

Paul Gray
Claremont Graduate School

User-Centered Requirements Analysis

Charles Fontaine Martin (Prentice Hall, Englewood Cliffs, N.J., 1988, 305 pp., \$30)

Charles Martin's book confronts requirements analysts with a fork in the road. One path is traditional, the other is radical. The traditional path gleams in the book's structure, material, and presentation. The radical path glimmers in reflective thought and awaits the reader's reaction (more about this later).

At the beginning, Martin boldly posts his book's hypothesis: "... important information system applications should be developed from clear, complete, and agreed-upon descriptions." For clarity, analysts must describe requirements in terms that both users and developers understand. For completeness, analysts must adequately define the application, the users it supports, and its performance. For agreement, analysts must gain approval of the requirements by users, sponsors, and developers.

Martin rests his case on two philosophical principles. First, users should see how they will interact with the system. Second, functions and data are equally important and should be developed in parallel. Further, he shows how his approach applies to the most common kinds of development cycles: classic, prototyping, and reusable code.

From this point, Martin presents his complete requirements analysis process. He divides requirements analysis into four phases: objectives analysis, user requirements generation, implementation assessment, and formal specification generation. He fully describes each phase with outlines, forms, and diagrams, and he thoughtfully augments his discussion with a sample application.

During objectives analysis, analysts transform the application charter into an application concept and a set of resource estimates. The application concept defines the application's objectives, users, and terminology. The terminology must include definitions of concepts leading to agreement by the requirements' audiences.

During user-requirements generation, analysts transform the application concept and resource estimates into a user-requirements document. The document includes the functional description, database description, and performance requirements agreed on by the user. Martin promotes his user-concept diagrams as the best framework for gaining agreement.

During implementation assessment, analysts develop risk assessments and an

implementation plan. The risk assessments help the sponsor match the user requirements to available resources. The results lead to the implementation plan.

During formal specification preparation, analysts transform the user speci-

Martin implicates requirements analysis with rhetoric, a form of reasoning foreign to scientists.

cations and implementation plan into the formal specification document.

Martin completes his discussion with a section on automated tools, requirements documentation, and project planning. He includes three appendixes containing a table of 41 commercial computer-aided software-engineering environments, a table of 10 common application-development environments, and a glossary. Each of the book's 18 chapters ends with a set of discussion questions and references. Martin accurately aims his text at graduate students in computer science or management information systems and practicing requirements analysts.

Traditionally, analysts link their work with science through system engineering. Engineers, like scientists, seek compelling reasoning through logic, mathematics, and empirical evidence. Compelling reasoning leads to scientific authority. However, analysts use competing methods, and no method compels adherence with the weight of scientific authority.

Martin provides that authority by treating system analysis in terms of style, audience, and agreement. But these are the subjects of persuasive reasoning — rhetoric — a subject traditionally scorned by scientists. Thus, perhaps unknowingly, Martin implicates requirements analysis with a form of reasoning foreign to scientists.

Martin's tacit implication of rhetoric opens a new, radical path of inquiry; analysts might ask how rhetoric can help solve their problems. For those daring to tread this road, *The New Rhetoric* (Chaim Perelman and L. Olbrechts-Tyteca, University of Notre Dame Press, 1969) provides a good starting point.

Irad D. Cole
Unisys

Data Compression: Methods and Theory

James A. Storer (Computer Science Press, Rockville, Md., 1988, 413 pp., \$44.95)

In this age of fax technology and growth, data compression has taken on more and more importance. Up to now, the teacher or practitioner has had sources like Gilbert Held's little handbook *Data Compression Techniques, and Applications: Hardware and Software Considerations* (John Wiley, 1983) and the survey of papers dealing with data compression edited by Robert M. Gray and Lee D. Davisson, *Data Compression* (Dowden, Hutchinson, & Ross, 1976). These are suitable as references; however, they are very seldom suitable for a course, seminar, or overview.

James A. Storer now gives us a survey of data compression, at least from the algorithmic perspective pointing toward software or firmware, from its beginning to the present. The book presents basic theory as well as algorithms that could be used by the scholar or the practicing computer scientist. Storer obviously wants to give the reader a foundation for understanding data compression, its methods, tools, and techniques. And he pours it on from start to finish, complete with rich appendixes of results, coding schemes, coding statistics needed by data compression algorithms, and even some of the algorithms themselves implemented in Pascal.

Storer assumes the reader has a reasonable level of mathematical sophistication and a general familiarity with the hardware, algorithms, and data structures, including knowledge of NP-completeness, automata theory, and undecidability. He dives right into the language and folklore of data compression. However, he also (kindly) provides extensive examples to illustrate the theory and an annotated bibliography at the end of each chapter. The curious or puzzled reader can pursue the bibliography to fill in the details of the terse, theoretical text and the results.

Storer moves swiftly from the introduction to data compression, its terminology, and its on-line and off-line techniques, to coding techniques including Huffman codes (mainly to provide a basis for comparing data compression techniques). He also surveys step codes, prefix codes, dynamic Huffman codes, Tunstall codes, arithmetic codes, and Shannon-Fano codes.

Then Storer presents practical methods for lossy and lossless compression and

on-line textual substitution of coded text for given initial text. This discussion includes separate chapters on sequential and parallel implementations (the latter is of particular interest to VLSI enthusiasts). Storer reintroduces off-line textual substitution to discuss the tractability of the encoding algorithms and develops results in NP-completeness and comparisons of greedy versus optimal algorithms.

Finally, Storer puts aside practical concerns in a summary chapter and uses the tools of abstract computational complexity theory to focus on the issue of what sorts of data compression can be achieved with unlimited computing

power and resources. Here the author looks at spaces consisting of compressed forms of strings and computer programs that recover the information that the compressed forms code.

The book is wonderful for the student or practitioner who needs to understand the application of courses in computability, data structures, and algorithms. It presents open problems and suggests further study by the reader.

The absence of formal exercises or problem sets is a drawback for using the book as a text. There are plenty of exercises involved in working out and

understanding the results presented, but the reader must write out and perform those exercises rather than be led through them. Hence, the book is very individualized, appropriate for a seminar or advanced undergraduate or graduate course. The book could also be used by a project team in preparation for a project requiring extensive and sophisticated knowledge of data compression techniques. This is a good book that fills a void for both the practitioner and scholar in computer science.

John W. Fendrich
Bradley University

NEW LITERATURE

Expert volumes. IEEE Press is offering two companion collections of selected reprints, *Principles of Expert Systems* (Order No. PCO2287, 464 pp., \$69.95, \$52.45 for IEEE members) and *Microcomputer-Based Expert Systems* (Order No. PCO2311, 352 pp., \$44.95, \$33.70 for IEEE members). Edited by Amar Gupta and Bandreddi E. Prasad, the books contain a total of 75 research papers and specially written tutorial and transitional material. Both books also may be ordered together under Order No. PCO2378 for \$103.40 (\$77.55 for IEEE members).

Contact the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

Updated handbook. The updated third edition of the *Electronics Engineers' Handbook* (ISBN 0-07-020982-0, 2,528 pp., \$89.50), edited by Donald G. Fink and Donald Christiansen, includes a new section on US and international standards and new material on digital and computer technology, including the Karmarkar algorithm, ROMs and PLAs, codecs, etc. More than 170 authorities contributed material in their areas of specialization.

Contact McGraw-Hill, 11 West 19th Street, New York, NY 10011, or call (800) 262-4729

VAX reference. The second edition of *Computer Programming and Architecture: The VAX* (ISBN 1-55558-015-7, 444 pp., \$38) by Henry M. Levy and Richard H. Eckhouse, Jr., has been designed as a textbook and a reference for students and computer professionals. It includes updated coverage of VAX assembly language programming, new chapters on multi- and microprocessing, expanded discussion of caches and translation buffers, and a new appendix on the Berkeley Unix assembler.

Contact Digital Press, 12 Crosby Drive, Bedford, MA 01730.

A look at standards. *Information Technology Standardization: Theory, Practice, and Organizations* (280 pp., \$24.95) by Carl F. Cargill examines the standardization process as a force for change in the industry. The first section reviews the history and current state of standards; the second section offers a primer on standards and standardization for providers, users, and others; and the third section examines standards organizations themselves. Publication is scheduled for spring 1989.

Contact Digital Press, 12 Crosby Drive, Bedford, MA 01730.

Development documentation. The central theme of the six-volume set

Software Development Documentation by Steve J. Ayer and Frank S. Patrinostro is that all software projects should be documented as the system is developed, not afterward. The complete set is available for \$260; individual volumes are \$49.50 each. Individual titles are: *Software Development Planning and Management Documents*, *Software Development Analysis Documentation*, *Software Development Design Documentation*, *Software Program and Test Documentation*, *Software Implementation Documentation*, and *Software Configuration Management Documentation*.

Contact Technical Communications Associates, 1250 Oakmead Pkwy., Suite 210, Sunnyvale, CA 94086, phone (408) 737-2665.

Object-oriented programming. *The OOPS Report*, a quarterly publication on object-oriented programming systems, provides information on the emerging technologies of Smalltalk and C++, novel applications, libraries, interfaces, rapid prototyping, reusable components, object-oriented databases, new design methodologies, and hardware. The subscription rate in the United Kingdom is £95; £105 elsewhere.

Contact The OOPS Report, PO Box 84, Plymouth, PL1 1SZ, England.

ADVERTISER INDEX

Aerospace Computer Security Applications Conference	71
Aptech Systems, Inc.	23
ASIC Seminar and Exhibit	103
CACI Products Company	1, Cover IV
Computer Society Membership	75, 113
Computer Society Ombudsman	75
Computer Society Press	77
Digital Equipment Corp.	106
Guidelines Software, Inc.	49
HCCA4	51
ICCAD '89	Cover II
Interactive Development Environments	61
Interactive Software Engineering Inc.	6-7
Module Generation and Silicon Compilation Workshop	99
New Jersey Institute of Technology	4
Parallel Processing Symposium	85
Real-Time Systems Symposium	79
Scientific and Engineering Software, Inc.	62
Software Engineering Conference	38-41
Systems Integration Conference	101
The Technical University of Darmstadt in Germany	4
University of Lowell	50
University of Nebraska-Lincoln	84
University of North Carolina at Charlotte	84
Visible Systems Corp.	24
Workstation Operating Systems Workshop	86
Zortech Inc.	5
Classified Advertising	114-123

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

Northern California and Pacific Northwest: Roy McDonald Assoc. Inc., 5915 Hollis St., Emeryville, CA 94608; (415) 653-2122.
Jim Olsen, P.O. Box 696, Hillsboro, OR 97123; (503) 640-2011.
Southern California and Mountain States: Richard C. Faust Co., 24050 Madison St., Suite 100, Torrance, CA 90505; (213) 373-9604.
Southwest: The House Co., 3000 Wesleyan, Suite 345, Houston, TX 77027; (713) 622-2868.
Midwest: Robert Acker & Assoc., 480 Central Ave., Northfield, IL 60093; (312) 441-6050.
East Coast: Atlantic Representative Group, 349 Maple Pl., Keyport, NJ 07735; (201) 739-1444.
New England: Arpin Associates, 40 Sterling St., Somerville, MA 02144; (617) 625-1777.
Europe: Heinz J. Görgens, Parkstrasse 8a, D-4054 Nettetal 1 - Hinsbeck (F.R.G.); phone: (0 21 53) 8 99 88; telex 841(17)2153310=HJG tlx d.
 Advertising Director: Dawn Peck. For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.
COMPUTER, 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; phone (714) 821-8380; fax (714) 821-4010.

PRODUCT INDEX

	RS#	Page#
BOARDS		
Board set	51	98
CIRCUITS		
Floating-point controller	50	98
IC announcements	120-132	100
Memory management unit	49	98
COMPONENTS		
Microsystem announcements	135-147	102
Silicon software	46	97
CONFERENCES & COURSES		
Aerospace Security Applications	—	71
ASIC	—	103
HCCA4	—	51
ICCAD '89	—	C.II
Module Generation and Silicon Compilation	—	99
Parallel Processing	—	85
Real-Time Systems	—	79
Software Engineering	—	38-41
System Integration	—	101
Workstation Operating Systems	—	86
DATA COMMUNICATIONS		
Display server	42	96
Network implementation	53	98
OTHER PRODUCTS & SERVICES		
Electronic security	45	97
Medical information system	52	98
PUBLISHERS		
CS membership	—	75, 113
CS ombudsman	—	75
CS Press	—	77
Document editor	41	96
RECRUITMENT		
Classified advertising	—	114-123
Technical professional	—	4, 50, 84, 106
SOFTWARE		
Basic	21	87
C compiler	1, 5, 23	5, 49, 92
DBMS software	47-48	98
Graphical shell	39	96
Matrix-handling program	3	23
Object-oriented environment	2	6-7
Pascal	22	89
Software tool	40	96
SYSTEMS		
Multiprocessor	30	94
Personal computer	36-38, 43	95, 96
Productivity tool	6, 7	61, 62
Project development tool	4	24
RISC workstation	34-35, 44	95, 97
Simulation package	—	1, C.IV
VAX workstation	31-32	94
TEST & MEASURING EQUIPMENT		
Computation server	33	95

COMPUTER

February 1989 issue (Card void after July 1989)

Please print or use peel-off address label.

Name _____

Title _____

Company _____

Address _____

City _____

State/ZIP _____

Country _____ Phone _____

Please send (circle those you want)

- | | |
|----------------------------|---------------------------------------|
| 201 Publications catalog | 203 Student membership information |
| 202 Membership information | 204 Application for senior membership |

Reader interest

(Circle what you liked;
add comments on the back)

- A InterViews
- B Interconnection Networks
- C Code Optimization
- D Ada Compiler Technology
- E ACM Task Force Report
- F Snowbird 88
- G Letters
- H Open Channel
- J Standards
- K Computer Society News
- L Update
- M Product Reviews
- N New Products
- P IC Announcements
- Q Microsystems
- R Conferences
- S Calendar
- T Call for Papers
- U Book Reviews
- V New Literature

Product inquiries

(circle numbers for products and advertisements
on which you want more information)

1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	191
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199
20	40	60	80	100	120	140	160	180	200

1

COMPUTER

February 1989 issue (Card void after July 1989)

Please print or use peel-off address label.

Name _____

Title _____

Company _____

Address _____

City _____

State/ZIP _____

Country _____ Phone _____

Please send (circle those you want)

- | | |
|----------------------------|---------------------------------------|
| 201 Publications catalog | 203 Student membership information |
| 202 Membership information | 204 Application for senior membership |

Reader interest

(Circle what you liked;
add comments on the back)

- A InterViews
- B Interconnection Networks
- C Code Optimization
- D Ada Compiler Technology
- E ACM Task Force Report
- F Snowbird 88
- G Letters
- H Open Channel
- J Standards
- K Computer Society News
- L Update
- M Product Reviews
- N New Products
- P IC Announcements
- Q Microsystems
- R Conferences
- S Calendar
- T Call for Papers
- U Book Reviews
- V New Literature

Product inquiries

(circle numbers for products and advertisements
on which you want more information)

1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	191
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199
20	40	60	80	100	120	140	160	180	200

2

COMPUTER

February 1989 issue (Card void after July 1989)

Please print or use peel-off address label.

Name _____

Title _____

Company _____

Address _____

City _____

State/ZIP _____

Country _____ Phone _____

Please send (circle those you want)

- | | |
|----------------------------|---------------------------------------|
| 201 Publications catalog | 203 Student membership information |
| 202 Membership information | 204 Application for senior membership |

Reader interest

(Circle what you liked;
add comments on the back)

- A InterViews
- B Interconnection Networks
- C Code Optimization
- D Ada Compiler Technology
- E ACM Task Force Report
- F Snowbird 88
- G Letters
- H Open Channel
- J Standards
- K Computer Society News
- L Update
- M Product Reviews
- N New Products
- P IC Announcements
- Q Microsystems
- R Conferences
- S Calendar
- T Call for Papers
- U Book Reviews
- V New Literature

Product inquiries

(circle numbers for products and advertisements
on which you want more information)

1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	191
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199
20	40	60	80	100	120	140	160	180	200

3

Editorial comments

I liked: _____

I disliked: _____

I would like to see: _____

For reader-service inquiries, see other side.



PO Box is for reader-service cards only.

PLACE
POSTAGE
HERE

COMPUTER

Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508



Editorial comments

I liked: _____

I disliked:

I would like to see:

For reader-service inquiries, see other side.



PO Box is for reader-service cards only.

PLACE
POSTAGE
HERE

COMPUTER

Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA



Editorial comments

I liked: _____

I disliked: _____

I would like to see: _____

For reader-service inquiries, see other side.



PO Box is for reader-service cards only.

PLACE
POSTAGE
HERE

COMPUTER

Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA





IEEE COMPUTER SOCIETY

A member society of the Institute of Electrical and Electronics Engineers, Inc.

Executive Committee

President: Kenneth R. Anderson*
Siemens Research & Technology
105 College Road East
Princeton, NJ 08540
(609) 734-6550

President-Elect: Helen M. Wood*
Past President: Edward A. Parrish, Jr.*

Vice Presidents

Conferences and Tutorials: Joseph E. Urban (1st VP)*

Technical Activities: Laurel V. Kaleda (2nd VP)*

Area Activities: Ned Kornfield†

Education: Gerald L. Engel†

Membership and Information: Barry W. Johnson†

Press Activities: Duncan H. Lawrie*

Publications: Sallie V. Sheppard*

Standards: Paul L. Borrill*

Secretary: Michael Evangelist*

Treasurer: Charles B. Silio†

Division V Director: Harriett Rigas†

Division VIII Director: Roy L. Russo†

Executive Director: T. Michael Elliott†

*voting member of the Board of Governors

†nonvoting member of the Board of Governors

Board of Governors

Term Expiring 1989:

Bill D. Carroll, Lansing (Chip) Hatfield,

Duncan H. Lawrie, David Pessel,

Susan L. Rosenbaum, Sallie V. Sheppard, Bruce Shriver, Harold S.

Stone, Akihiko Yamada, Marshall C. Yovits

Term Expiring 1990:

Vishwani Agrawal, Mario R. Barbacci,

Ming T. (Mike) Liu, Yale N. Patt, Donald E. Thomas, Benjamin W.

Wah, Ronald Waxman

Term Expiring 1991:

P. Bruce Berra, Paul L. Borrill, Michael Evangelist,

Ted Lewis, Raymond E. Miller,

Earl E. Swartzlander, Jr., Thomas W. Williams

Next Board Meeting

March 3, 1989: 8:30 a.m.

Cathedral Hill Hotel, San Francisco

Senior Staff

Executive Director: T. Michael Elliott

Editor and Publisher: H. True Seaborn

Director, Conferences: Anne Marie Kelly

Director, Finance and Administration: Tod S. Heisler

Assistant to the Executive Director: Violet S. Doan

Computer Society Offices

Headquarters Office

1730 Massachusetts Ave. NW

Washington, DC 20036-1903

Phone (202) 371-0101

Telex: 7180250437 IEEE COMPSO

Fax: (202) 728-9614

Publications Office

10662 Los Vaqueros Cir.

Los Alamitos, CA 90720-2578

Membership and General Information: (714) 821-8380

Publication Orders: (800) 272-6657

Fax: (714) 821-4010

European Office

13, Ave. de L'Aquilon

B-1200 Brussels, Belgium

Phone: 32 (2) 770-21-98

Telex: 25387 AVVALB

Fax: 32 (2) 770-85-05

Asian Office

Ooshima Building

2-19-1 Minami-Aoyama, Minato-ku

Tokyo 107, Japan

Phone: 81 (3) 408-3118

Fax: 81 (3) 408-3553

Use the Reader Service Card to obtain information on:

- Membership application—student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #210
- Standards working groups list #195
- Compmail+ international electronic mail/database brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures—student/regular #193
- Student scholarship information #192
- Awards description/nomination forms #198
- Volunteer leaders/staff directory #196
- IEEE senior member application #204

Purpose

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 97,000 members worldwide, and provides a wide range of services to members and nonmembers.

Membership

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others seriously interested in the computer field.

Publications and Activities

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and new products.

Periodicals. The society publishes six magazines and four research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standards Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. Over 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

European and Asian Offices

Payments for Computer Society membership and publication orders are accepted by checks or bank transfer in Belgian, British, German, Japanese, Swiss, or US currency, or by American Express, Eurocard, MasterCard, or Visa credit cards.

Ombudsman

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

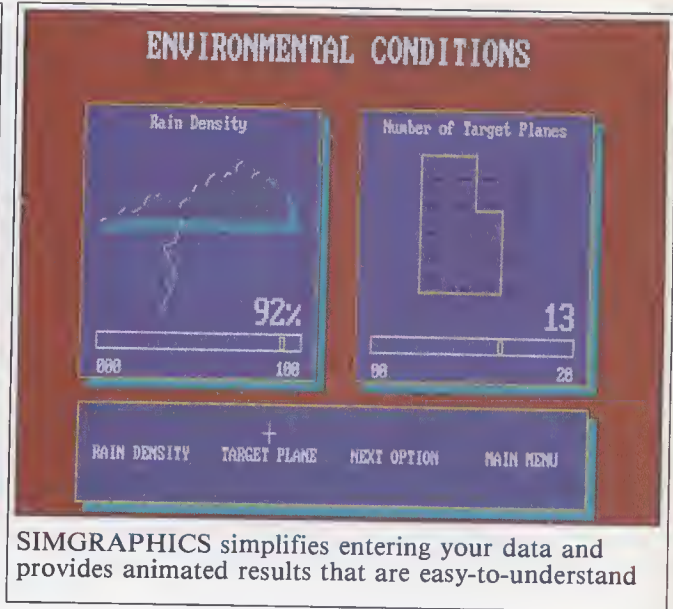
SIMULATION BREAKTHROUGH

BEFORE

TEST RETS 0	SSL(1)	-1.67	X	-1.73	SSL(2)	4.77	V	4.82	V1	4.00
TEST RETS 0	SSL(1)	-1.67	X	-1.70	SSL(2)	4.77	V	4.80	V1	4.00
TEST RETS 0	SSL(1)	-1.70	X	-1.73	SSL(2)	4.80	V	4.82	V1	4.00
PAGE 6										
TIME	X	Y	SPEED		THETA	DX,DT		DY,DT		
TEST RETS 0	SSL(1)	-1.73	X	-1.83	SSL(2)	4.82	V	4.90	V1	4.00
TEST RETS 0	SSL(1)	-1.73	X	-1.70	SSL(2)	4.82	V	4.86	V1	4.00
TEST RETS 0	SSL(1)	-1.73	X	-1.75	SSL(2)	4.82	V	4.84	V1	4.00
TEST RETS 0	SSL(1)	-1.75	X	-1.70	SSL(2)	4.84	V	4.86	V1	4.00
TEST RETS 0	SSL(1)	-1.70	X	-1.85	SSL(2)	4.86	V	4.92	V1	4.00
TEST RETS 0	SSL(1)	-1.70	X	-1.81	SSL(2)	4.86	V	4.89	V1	4.00
TEST RETS 0	SSL(1)	-1.70	X	-1.79	SSL(2)	4.86	V	4.88	V1	4.00
TEST RETS 0	SSL(1)	-1.79	X	-1.81	SSL(2)	4.89	V	4.89	V1	4.00
TEST RETS 0	SSL(1)	-1.81	X	-1.87	SSL(2)	4.89	V	4.94	V1	4.00
TEST RETS 0	SSL(1)	-1.81	X	-1.84	SSL(2)	4.89	V	4.92	V1	4.00
TEST RETS 0	SSL(1)	-1.84	X	-1.87	SSL(2)	4.92	V	4.94	V1	4.00
TEST RETS 0	SSL(1)	-1.87	X	-1.90	SSL(2)	4.94	V	4.96	V1	4.00
TEST RETS 0	SSL(1)	-1.90	X	-1.93	SSL(2)	4.96	V	4.98	V1	4.00
.130000 -1.93049 4.98318 051.90510 .04230 -48.6560 36.8210										

Difficult to understand results

AFTER



SIMGRAPHICS simplifies entering your data and provides animated results that are easy-to-understand

Announcing new SIMSCRIPT II.5 with SIMGRAPHICS

Simulation models are now easier to build
--results are easier to understand

Now you can provide the users of your SIMSCRIPT II.5 models with SIMGRAPHICS™ --graphical input and animation.

Results are easy to understand--an animated picture, histograms, pie charts and plots.

Because your results are understood, your recommendations are more likely to be acted upon.

Simulation simplified

SIMSCRIPT II.5 gives you a compact English-like language. Your simulation program reads like a description of the system you are studying. Animation is easy.

Your model development, check-out, modification and enhancement are greatly simplified.

Many successful applications

SIMSCRIPT II.5® is a well established, standardized, and widely used language with proven software support.

Typical applications include: military planning, manufacturing, communications, logistics, and transportation.

Free trial and training offer

The free trial contains everything you need to try SIMSCRIPT II.5 on your computer.

You may develop your model or modify one of ours.

Try the SIMSCRIPT II.5 language, the quality and timeliness of our support, the accuracy of our documentation and the facilities for error-checking--**everything you need for a successful project.**

No cost, no obligation.

Act now--free training offer

For a limited time we also include free training. Typical applications are demonstrated.

Call today to avoid disappointment--class size is limited.

For immediate information

Call Doug Dittrich at (619) 457-9681. In the UK, call Richard Eve on (01) 528-7980.

With SIMSCRIPT II.5 you get results sooner and they are better understood.

Rush information on SIMSCRIPT II.5 with SIM- GRAPHICS free trial and training

Free trial--learn the reasons for the broad and growing popularity of SIMSCRIPT II.5.

No cost, no obligation.

Limited offer--return the coupon today and we will include one free course enrollment worth \$950.

☐ Send information on your Special University Offer.

Name _____

Organization _____

Address _____

City _____

State _____

Zip _____

Telephone _____

Computer _____

Operating System _____

Return to:

IEEE COMP

CACI Products Company
3344 North Torrey Pines Court
La Jolla, California 92037
Or, better yet, call Doug Dittrich at
(619) 457-9681, FAX (619) 457-1184

In the UK

CACI Products Division
Regent House, 89 Kingsway
London WC2B 6RH, United Kingdom
Call Richard Eve on (01) 528-7980

SIMSCRIPT II.5, and SIMGRAPHICS are registered trademarks and service marks of CACI, INC.
©1989 CACI, INC.